Boost.Asio

Christopher Kohlhoff

Copyright © 2003-2012 Christopher M. Kohlhoff

Distributed under the Boost Software License, Version 1.0. (See accompanying file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)

Boost.Asio is a cross-platform C++ library for network and low-level I/O programming that provides developers with a consistent asynchronous model using a modern C++ approach.

Overview	An overview of the features included in Boost.Asio, plus rationale and design information.
Using Boost.Asio	How to use Boost.Asio in your applications. Includes information on library dependencies and supported platforms.
Tutorial	A tutorial that introduces the fundamental concepts required to use Boost.Asio, and shows how to use Boost.Asio to develop simple client and server programs.
Examples	Examples that illustrate the use of Boost.Asio in more complex applications.
Reference	Detailed class and function reference.
Revision History	Log of Boost.Asio changes made in each Boost release.
Index	Book-style text index of Boost.Asio documentation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Overview

- Rationale
- Core Concepts and Functionality
  - Basic Boost.Asio Anatomy
  - The Proactor Design Pattern: Concurrency Without Threads
  - Threads and Boost.Asio
  - Strands: Use Threads Without Explicit Locking
  - Buffers
  - Streams, Short Reads and Short Writes
  - Reactor-Style Operations
  - Line-Based Operations
  - Custom Memory Allocation
  - Handler Tracking
- Networking
  - TCP, UDP and ICMP
  - Socket Iostreams
  - The BSD Socket API and Boost.Asio
- Timers
- Serial Ports
- Signal Handling
- POSIX-Specific Functionality
  - UNIX Domain Sockets
  - Stream-Oriented File Descriptors
  - Fork
- Windows-Specific Functionality
  - Stream-Oriented HANDLEs
  - Random-Access HANDLEs
  - Object HANDLEs
- SSL
- C++ 2011 Support
  - Movable I/O Objects

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- Movable Handlers
- Variadic Templates
- Array Container
- Atomics
- Shared Pointers
- Chrono

- Platform-Specific Implementation Notes

Rationale

Most programs interact with the outside world in some way, whether it be via a file, a network, a serial cable, or the console. Sometimes, as is the case with networking, individual I/O operations can take a long time to complete. This poses particular challenges to application development.

Boost.Asio provides the tools to manage these long running operations, without requiring programs to use concurrency models based on threads and explicit locking.

The Boost.Asio library is intended for programmers using C++ for systems programming, where access to operating system functionality such as networking is often required. In particular, Boost.Asio addresses the following goals:

- **Portability.** The library should support a range of commonly used operating systems, and provide consistent behaviour across these operating systems.
- **Scalability.** The library should facilitate the development of network applications that scale to thousands of concurrent connections. The library implementation for each operating system should use the mechanism that best enables this scalability.
- **Efficiency.** The library should support techniques such as scatter-gather I/O, and allow programs to minimise data copying.
- **Model concepts from established APIs, such as BSD sockets.** The BSD socket API is widely implemented and understood, and is covered in much literature. Other programming languages often use a similar interface for networking APIs. As far as is reasonable, Boost.Asio should leverage existing practice.
- **Ease of use.** The library should provide a lower entry barrier for new users by taking a toolkit, rather than framework, approach. That is, it should try to minimise the up-front investment in time to just learning a few basic rules and guidelines. After that, a library user should only need to understand the specific functions that are being used.
- **Basis for further abstraction.** The library should permit the development of other libraries that provide higher levels of abstraction. For example, implementations of commonly used protocols such as HTTP.

Although Boost.Asio started life focused primarily on networking, its concepts of asynchronous I/O have been extended to include other operating system resources such as serial ports, file descriptors, and so on.

Core Concepts and Functionality

- Basic Boost.Asio Anatomy
- The Proactor Design Pattern: Concurrency Without Threads
- Threads and Boost.Asio
- Strands: Use Threads Without Explicit Locking
- Buffers

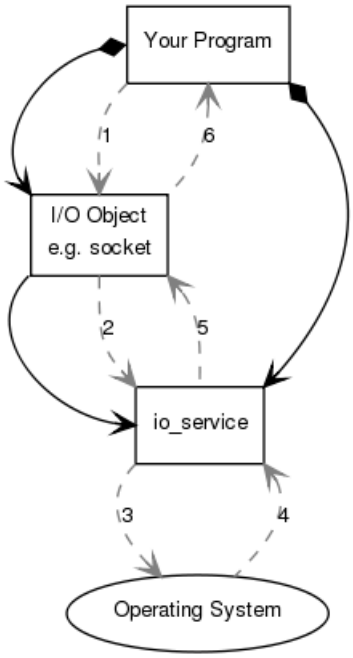
XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- Streams, Short Reads and Short Writes
- Reactor-Style Operations
- Line-Based Operations
- Custom Memory Allocation
- Handler Tracking

Basic Boost.Asio Anatomy

Boost.Asio may be used to perform both synchronous and asynchronous operations on I/O objects such as sockets. Before using Boost.Asio it may be useful to get a conceptual picture of the various parts of Boost.Asio, your program, and how they work together.

As an introductory example, let's consider what happens when you perform a connect operation on a socket. We shall start by examining synchronous operations.



Your program will have at least one **io_service** object. The **io_service** represents **your program**'s link to the **operating system**'s I/O services.

```
boost::asio::io_service io_service;
```

To perform I/O operations **your program** will need an **I/O object** such as a TCP socket:

```
boost::asio::ip::tcp::socket socket(io_service);
```

When a synchronous connect operation is performed, the following sequence of events occurs:

1. **Your program** initiates the connect operation by calling the **I/O object**:

```
socket.connect(server_endpoint);
```

2. The **I/O object** forwards the request to the **io_service**.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

3. The **io_service** calls on the **operating system** to perform the connect operation.

4. The **operating system** returns the result of the operation to the **io_service**.

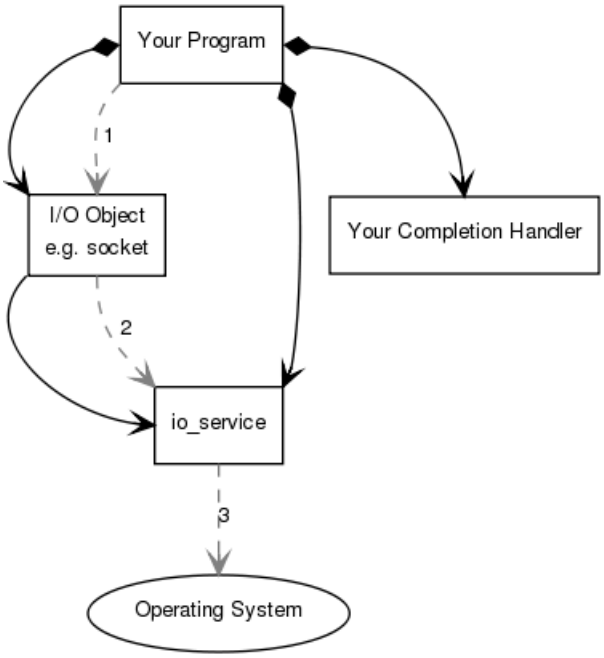
5. The **io_service** translates any error resulting from the operation into an object of type `boost::system::error_code`. An `error_code` may be compared with specific values, or tested as a boolean (where a `false` result means that no error occurred). The result is then forwarded back up to the **I/O object**.

6. The **I/O object** throws an exception of type `boost::system::system_error` if the operation failed. If the code to initiate the operation had instead been written as:

```
boost::system::error_code ec;
socket.connect(server_endpoint, ec);
```

then the `error_code` variable `ec` would be set to the result of the operation, and no exception would be thrown.

When an asynchronous operation is used, a different sequence of events occurs.



1. **Your program** initiates the connect operation by calling the **I/O object**:

```
socket.async_connect(server_endpoint, your_completion_handler);
```

where `your_completion_handler` is a function or function object with the signature:

```
void your_completion_handler(const boost::system::error_code& ec);
```

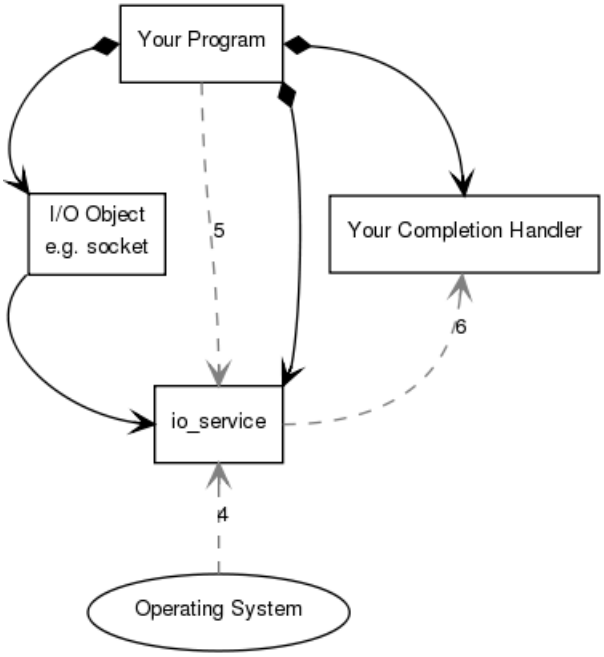
The exact signature required depends on the asynchronous operation being performed. The reference documentation indicates the appropriate form for each operation.

2. The **I/O object** forwards the request to the **io_service**.

3. The **io_service** signals to the **operating system** that it should start an asynchronous connect.

Time passes. (In the synchronous case this wait would have been contained entirely within the duration of the connect operation.)

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/



4. The **operating system** indicates that the connect operation has completed by placing the result on a queue, ready to be picked up by the **io_service**.

5. **Your program** must make a call to `io_service::run()` (or to one of the similar **io_service** member functions) in order for the result to be retrieved. A call to `io_service::run()` blocks while there are unfinished asynchronous operations, so you would typically call it as soon as you have started your first asynchronous operation.

6. While inside the call to `io_service::run()`, the **io_service** dequeues the result of the operation, translates it into an `error_code`, and then passes it to **your completion handler**.

This is a simplified picture of how Boost.Asio operates. You will want to delve further into the documentation if your needs are more advanced, such as extending Boost.Asio to perform other types of asynchronous operations.

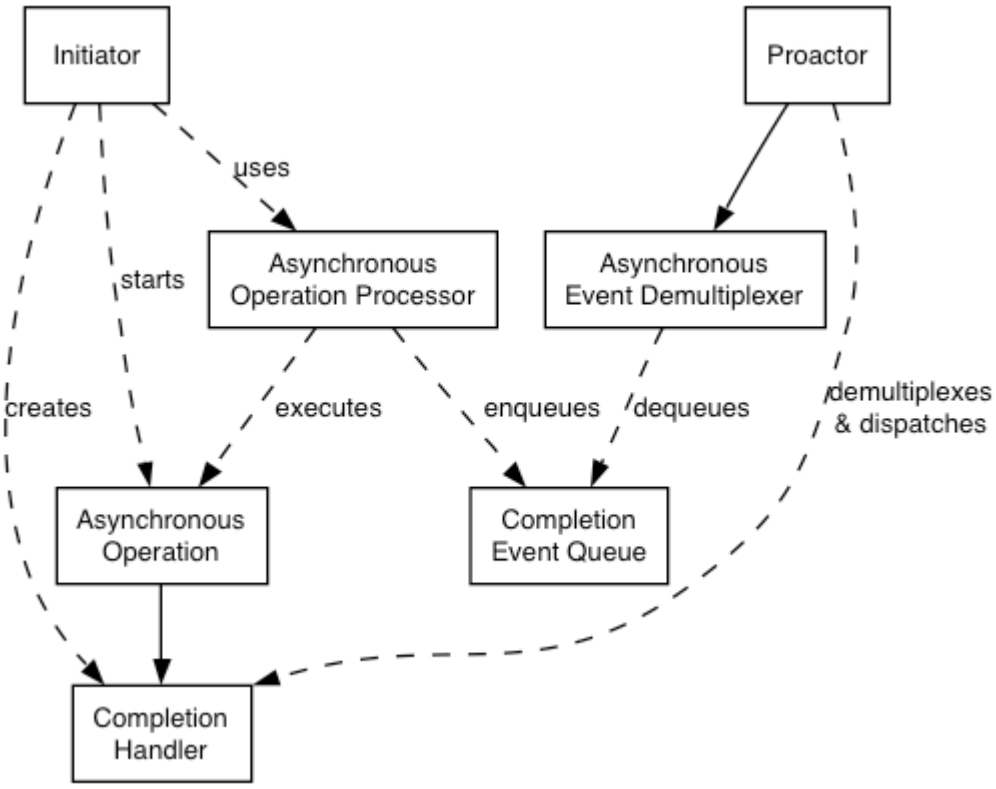
The Proactor Design Pattern: Concurrency Without Threads

The Boost.Asio library offers side-by-side support for synchronous and asynchronous operations. The asynchronous support is based on the Proactor design pattern [POSA2]. The advantages and disadvantages of this approach, when compared to a synchronous-only or Reactor approach, are outlined below.

Proactor and Boost.Asio

Let us examine how the Proactor design pattern is implemented in Boost.Asio, without reference to platform-specific details.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/



Proactor design pattern (adapted from [POSA2])

— Asynchronous Operation

Defines an operation that is executed asynchronously, such as an asynchronous read or write on a socket.

— Asynchronous Operation Processor

Executes asynchronous operations and queues events on a completion event queue when operations complete. From a high-level point of view, services like `stream_socket_service` are asynchronous operation processors.

— Completion Event Queue

Buffers completion events until they are dequeued by an asynchronous event demultiplexer.

— Completion Handler

Processes the result of an asynchronous operation. These are function objects, often created using `boost::bind`.

— Asynchronous Event Demultiplexer

Blocks waiting for events to occur on the completion event queue, and returns a completed event to its caller.

— Proactor

Calls the asynchronous event demultiplexer to dequeue events, and dispatches the completion handler (i.e. invokes the function object) associated with the event. This abstraction is represented by the `io_service` class.

— Initiator

Application-specific code that starts asynchronous operations. The initiator interacts with an asynchronous operation processor via a high-level interface such as `basic_stream_socket`, which in turn delegates to a service like `stream_socket_service`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Implementation Using Reactor

On many platforms, Boost.Asio implements the Proactor design pattern in terms of a Reactor, such as `select`, `epoll` or `kqueue`. This implementation approach corresponds to the Proactor design pattern as follows:

— Asynchronous Operation Processor

> A reactor implemented using `select`, `epoll` or `kqueue`. When the reactor indicates that the resource is ready to perform the operation, the processor executes the asynchronous operation and enqueues the associated completion handler on the completion event queue.

— Completion Event Queue

> A linked list of completion handlers (i.e. function objects).

— Asynchronous Event Demultiplexer

> This is implemented by waiting on an event or condition variable until a completion handler is available in the completion event queue.

Implementation Using Windows Overlapped I/O

On Windows NT, 2000 and XP, Boost.Asio takes advantage of overlapped I/O to provide an efficient implementation of the Proactor design pattern. This implementation approach corresponds to the Proactor design pattern as follows:

— Asynchronous Operation Processor

> This is implemented by the operating system. Operations are initiated by calling an overlapped function such as `AcceptEx`.

— Completion Event Queue

> This is implemented by the operating system, and is associated with an I/O completion port. There is one I/O completion port for each `io_service` instance.

— Asynchronous Event Demultiplexer

> Called by Boost.Asio to dequeue events and their associated completion handlers.

Advantages

— Portability.

> Many operating systems offer a native asynchronous I/O API (such as overlapped I/O on *Windows*) as the preferred option for developing high performance network applications. The library may be implemented in terms of native asynchronous I/O. However, if native support is not available, the library may also be implemented using synchronous event demultiplexors that typify the Reactor pattern, such as *POSIX* `select()`.

— Decoupling threading from concurrency.

> Long-duration operations are performed asynchronously by the implementation on behalf of the application. Consequently applications do not need to spawn many threads in order to increase concurrency.

— Performance and scalability.

> Implementation strategies such as thread-per-connection (which a synchronous-only approach would require) can degrade system performance, due to increased context switching, synchronisation and data movement among CPUs. With asynchronous operations it is possible to avoid the cost of context switching by minimising the number of operating system threads — typically a limited resource — and only activating the logical threads of control that have events to process.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

— Simplified application synchronisation.

Asynchronous operation completion handlers can be written as though they exist in a single-threaded environment, and so application logic can be developed with little or no concern for synchronisation issues.

— Function composition.

Function composition refers to the implementation of functions to provide a higher-level operation, such as sending a message in a particular format. Each function is implemented in terms of multiple calls to lower-level read or write operations.

For example, consider a protocol where each message consists of a fixed-length header followed by a variable length body, where the length of the body is specified in the header. A hypothetical read_message operation could be implemented using two lower-level reads, the first to receive the header and, once the length is known, the second to receive the body.

To compose functions in an asynchronous model, asynchronous operations can be chained together. That is, a completion handler for one operation can initiate the next. Starting the first call in the chain can be encapsulated so that the caller need not be aware that the higher-level operation is implemented as a chain of asynchronous operations.

The ability to compose new operations in this way simplifies the development of higher levels of abstraction above a networking library, such as functions to support a specific protocol.

Disadvantages

— Program complexity.

It is more difficult to develop applications using asynchronous mechanisms due to the separation in time and space between operation initiation and completion. Applications may also be harder to debug due to the inverted flow of control.

— Memory usage.

Buffer space must be committed for the duration of a read or write operation, which may continue indefinitely, and a separate buffer is required for each concurrent operation. The Reactor pattern, on the other hand, does not require buffer space until a socket is ready for reading or writing.

References

[POSA2] D. Schmidt et al, *Pattern Oriented Software Architecture, Volume 2*. Wiley, 2000.

Threads and Boost.Asio

Thread Safety

In general, it is safe to make concurrent use of distinct objects, but unsafe to make concurrent use of a single object. However, types such as `io_service` provide a stronger guarantee that it is safe to use a single object concurrently.

Thread Pools

Multiple threads may call `io_service::run()` to set up a pool of threads from which completion handlers may be invoked. This approach may also be used with `io_service::post()` to use a means to perform any computational tasks across a thread pool.

Note that all threads that have joined an `io_service`'s pool are considered equivalent, and the `io_service` may distribute work across them in an arbitrary fashion.

Internal Threads

The implementation of this library for a particular platform may make use of one or more internal threads to emulate asynchronicity. As far as possible, these threads must be invisible to the library user. In particular, the threads:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- must not call the user's code directly; and
- must block all signals.

This approach is complemented by the following guarantee:

- Asynchronous completion handlers will only be called from threads that are currently calling `io_service::run()`.

Consequently, it is the library user's responsibility to create and manage all threads to which the notifications will be delivered.

The reasons for this approach include:

- By only calling `io_service::run()` from a single thread, the user's code can avoid the development complexity associated with synchronisation. For example, a library user can implement scalable servers that are single-threaded (from the user's point of view).
- A library user may need to perform initialisation in a thread shortly after the thread starts and before any other application code is executed. For example, users of Microsoft's COM must call `CoInitializeEx` before any other COM operations can be called from that thread.
- The library interface is decoupled from interfaces for thread creation and management, and permits implementations on platforms where threads are not available.

See Also

io_service.

Strands: Use Threads Without Explicit Locking

A strand is defined as a strictly sequential invocation of event handlers (i.e. no concurrent invocation). Use of strands allows execution of code in a multithreaded program without the need for explicit locking (e.g. using mutexes).

Strands may be either implicit or explicit, as illustrated by the following alternative approaches:

- Calling io_service::run() from only one thread means all event handlers execute in an implicit strand, due to the io_service's guarantee that handlers are only invoked from inside run().
- Where there is a single chain of asynchronous operations associated with a connection (e.g. in a half duplex protocol implementation like HTTP) there is no possibility of concurrent execution of the handlers. This is an implicit strand.
- An explicit strand is an instance of `io_service::strand`. All event handler function objects need to be wrapped using `io_service::strand::wrap()` or otherwise posted/dispatched through the `io_service::strand` object.

In the case of composed asynchronous operations, such as `async_read()` or `async_read_until()`, if a completion handler goes through a strand, then all intermediate handlers should also go through the same strand. This is needed to ensure thread safe access for any objects that are shared between the caller and the composed operation (in the case of `async_read()` it's the socket, which the caller can close() to cancel the operation). This is done by having hook functions for all intermediate handlers which forward the calls to the customisable hook associated with the final handler:

```
struct my_handler
{
  void operator()() { ... }
};

template<class F>
void asio_handler_invoke(F f, my_handler*)
{
  // Do custom invocation here.
  // Default implementation calls f();
}
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The `io_service::strand::wrap()` function creates a new completion handler that defines `asio_handler_invoke` so that the function object is executed through the strand.

See Also

io_service::strand, tutorial Timer.5, HTTP server 3 example.

Buffers

Fundamentally, I/O involves the transfer of data to and from contiguous regions of memory, called buffers. These buffers can be simply expressed as a tuple consisting of a pointer and a size in bytes. However, to allow the development of efficient network applications, Boost.Asio includes support for scatter-gather operations. These operations involve one or more buffers:

- A scatter-read receives data into multiple buffers.
- A gather-write transmits multiple buffers.

Therefore we require an abstraction to represent a collection of buffers. The approach used in Boost.Asio is to define a type (actually two types) to represent a single buffer. These can be stored in a container, which may be passed to the scatter-gather operations.

In addition to specifying buffers as a pointer and size in bytes, Boost.Asio makes a distinction between modifiable memory (called mutable) and non-modifiable memory (where the latter is created from the storage for a const-qualified variable). These two types could therefore be defined as follows:

```
typedef std::pair<void*, std::size_t> mutable_buffer;
typedef std::pair<const void*, std::size_t> const_buffer;
```

Here, a mutable_buffer would be convertible to a const_buffer, but conversion in the opposite direction is not valid.

However, Boost.Asio does not use the above definitions as-is, but instead defines two classes: `mutable_buffer` and `const_buffer`. The goal of these is to provide an opaque representation of contiguous memory, where:

- Types behave as std::pair would in conversions. That is, a `mutable_buffer` is convertible to a `const_buffer`, but the opposite conversion is disallowed.
- There is protection against buffer overruns. Given a buffer instance, a user can only create another buffer representing the same range of memory or a sub-range of it. To provide further safety, the library also includes mechanisms for automatically determining the size of a buffer from an array, `boost::array` or `std::vector` of POD elements, or from a `std::string`.
- Type safety violations must be explicitly requested using the `buffer_cast` function. In general an application should never need to do this, but it is required by the library implementation to pass the raw memory to the underlying operating system functions.

Finally, multiple buffers can be passed to scatter-gather operations (such as read() or write()) by putting the buffer objects into a container. The `MutableBufferSequence` and `ConstBufferSequence` concepts have been defined so that containers such as `std::vector`, `std::list`, `std::vector` or `boost::array` can be used.

Streambuf for Integration with Iostreams

The class `boost::asio::basic_streambuf` is derived from `std::basic_streambuf` to associate the input sequence and output sequence with one or more objects of some character array type, whose elements store arbitrary values. These character array objects are internal to the streambuf object, but direct access to the array elements is provided to permit them to be used with I/O operations, such as the send or receive operations of a socket:

- The input sequence of the streambuf is accessible via the data() member function. The return type of this function meets the `ConstBufferSequence` requirements.
- The output sequence of the streambuf is accessible via the prepare() member function. The return type of this function meets the `MutableBufferSequence` requirements.
- Data is transferred from the front of the output sequence to the back of the input sequence by calling the commit() member function.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- Data is removed from the front of the input sequence by calling the consume() member function.

The streambuf constructor accepts a `size_t` argument specifying the maximum of the sum of the sizes of the input sequence and output sequence. Any operation that would, if successful, grow the internal data beyond this limit will throw a `std::length_error` exception.

Bytewise Traversal of Buffer Sequences

The `buffers_iterator<>` class template allows buffer sequences (i.e. types meeting `MutableBufferSequence` or `ConstBufferSequence` requirements) to be traversed as though they were a contiguous sequence of bytes. Helper functions called buffers_begin() and buffers_end() are also provided, where the buffers_iterator<> template parameter is automatically deduced.

As an example, to read a single line from a socket and into a `std::string`, you may write:

```
boost::asio::streambuf sb;
...
std::size_t n = boost::asio::read_until(sock, sb, '\n');
boost::asio::streambuf::const_buffers_type bufs = sb.data();
std::string line(
    boost::asio::buffers_begin(bufs),
    boost::asio::buffers_begin(bufs) + n);
```

Buffer Debugging

Some standard library implementations, such as the one that ships with Microsoft Visual C++ 8.0 and later, provide a feature called iterator debugging. What this means is that the validity of iterators is checked at runtime. If a program tries to use an iterator that has been invalidated, an assertion will be triggered. For example:

```
std::vector<int> v(1)
std::vector<int>::iterator i = v.begin();
v.clear(); // invalidates iterators
*i = 0; // assertion!
```

Boost.Asio takes advantage of this feature to add buffer debugging. Consider the following code:

```
void dont_do_this()
{
 std::string msg = "Hello, world!";
 boost::asio::async_write(sock, boost::asio::buffer(msg), my_handler);
}
```

When you call an asynchronous read or write you need to ensure that the buffers for the operation are valid until the completion handler is called. In the above example, the buffer is the `std::string` variable `msg`. This variable is on the stack, and so it goes out of scope before the asynchronous operation completes. If you're lucky then the application will crash, but random failures are more likely.

When buffer debugging is enabled, Boost.Asio stores an iterator into the string until the asynchronous operation completes, and then dereferences it to check its validity. In the above example you would observe an assertion failure just before Boost.Asio tries to call the completion handler.

This feature is automatically made available for Microsoft Visual Studio 8.0 or later and for GCC when `_GLIBCXX_DEBUG` is defined. There is a performance cost to this checking, so buffer debugging is only enabled in debug builds. For other compilers it may be enabled by defining `BOOST_ASIO_ENABLE_BUFFER_DEBUGGING`. It can also be explicitly disabled by defining `BOOST_ASIO_DISABLE_BUFFER_DEBUGGING`.

See Also

buffer, buffers_begin, buffers_end, buffers_iterator, const_buffer, const_buffers_1, mutable_buffer, mutable_buffers_1, streambuf, ConstBufferSequence, MutableBufferSequence, buffers example.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Streams, Short Reads and Short Writes

Many I/O objects in Boost.Asio are stream-oriented. This means that:

- There are no message boundaries. The data being transferred is a continuous sequence of bytes.
- Read or write operations may transfer fewer bytes than requested. This is referred to as a short read or short write.

Objects that provide stream-oriented I/O model one or more of the following type requirements:

- `SyncReadStream`, where synchronous read operations are performed using a member function called `read_some()`.
- `AsyncReadStream`, where asynchronous read operations are performed using a member function called `async_read_some()`.
- `SyncWriteStream`, where synchronous write operations are performed using a member function called `write_some()`.
- `AsyncWriteStream`, where synchronous write operations are performed using a member function called `async_write_some()`.

Examples of stream-oriented I/O objects include `ip::tcp::socket`, `ssl::stream<>`, `posix::stream_descriptor`, `windows::stream_handle`, etc.

Programs typically want to transfer an exact number of bytes. When a short read or short write occurs the program must restart the operation, and continue to do so until the required number of bytes has been transferred. Boost.Asio provides generic functions that do this automatically: `read()`, `async_read()`, `write()` and `async_write()`.

Why EOF is an Error

- The end of a stream can cause `read`, `async_read`, `read_until` or `async_read_until` functions to violate their contract. E.g. a read of N bytes may finish early due to EOF.
- An EOF error may be used to distinguish the end of a stream from a successful read of size 0.

See Also

async_read(), async_write(), read(), write(), AsyncReadStream, AsyncWriteStream, SyncReadStream, SyncWriteStream.

Reactor-Style Operations

Sometimes a program must be integrated with a third-party library that wants to perform the I/O operations itself. To facilitate this, Boost.Asio includes a `null_buffers` type that can be used with both read and write operations. A `null_buffers` operation doesn't return until the I/O object is "ready" to perform the operation.

As an example, to perform a non-blocking read something like the following may be used:

```
ip::tcp::socket socket(my_io_service);
...
socket.non_blocking(true);
...
socket.async_read_some(null_buffers(), read_handler);
...
void read_handler(boost::system::error_code ec)
{
  if (!ec)
  {
    std::vector<char> buf(socket.available());
    socket.read_some(buffer(buf));
  }
}
```

These operations are supported for sockets on all platforms, and for the POSIX stream-oriented descriptor classes.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

See Also

null_buffers, basic_socket::non_blocking(), basic_socket::native_non_blocking(), nonblocking example.

Line-Based Operations

Many commonly-used internet protocols are line-based, which means that they have protocol elements that are delimited by the character sequence `"\r\n"`. Examples include HTTP, SMTP and FTP. To more easily permit the implementation of line-based protocols, as well as other protocols that use delimiters, Boost.Asio includes the functions `read_until()` and `async_read_until()`.

The following example illustrates the use of `async_read_until()` in an HTTP server, to receive the first line of an HTTP request from a client:

```
class http_connection
{
  ...

  void start()
  {
    boost::asio::async_read_until(socket_, data_, "\r\n",
        boost::bind(&http_connection::handle_request_line, this, _1));
  }

  void handle_request_line(boost::system::error_code ec)
  {
    if (!ec)
    {
      std::string method, uri, version;
      char sp1, sp2, cr, lf;
      std::istream is(&data_);
      is.unsetf(std::ios_base::skipws);
      is >> method >> sp1 >> uri >> sp2 >> version >> cr >> lf;
      ...
    }
  }

  ...

  boost::asio::ip::tcp::socket socket_;
  boost::asio::streambuf data_;
};
```

The `streambuf` data member serves as a place to store the data that has been read from the socket before it is searched for the delimiter. It is important to remember that there may be additional data *after* the delimiter. This surplus data should be left in the `streambuf` so that it may be inspected by a subsequent call to `read_until()` or `async_read_until()`.

The delimiters may be specified as a single `char`, a `std::string` or a `boost::regex`. The `read_until()` and `async_read_until()` functions also include overloads that accept a user-defined function object called a match condition. For example, to read data into a streambuf until whitespace is encountered:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef boost::asio::buffers_iterator<
    boost::asio::streambuf::const_buffers_type> iterator;

std::pair<iterator, bool>
match_whitespace(iterator begin, iterator end)
{
  iterator i = begin;
  while (i != end)
    if (std::isspace(*i++))
      return std::make_pair(i, true);
  return std::make_pair(i, false);
}
...
boost::asio::streambuf b;
boost::asio::read_until(s, b, match_whitespace);
```

To read data into a streambuf until a matching character is found:

```
class match_char
{
public:
  explicit match_char(char c) : c_(c) {}

  template <typename Iterator>
  std::pair<Iterator, bool> operator()(
      Iterator begin, Iterator end) const
  {
    Iterator i = begin;
    while (i != end)
      if (c_ == *i++)
        return std::make_pair(i, true);
    return std::make_pair(i, false);
  }

private:
  char c_;
};

namespace boost { namespace asio {
  template <> struct is_match_condition<match_char>
    : public boost::true_type {};
} } // namespace boost::asio
...
boost::asio::streambuf b;
boost::asio::read_until(s, b, match_char('a'));
```

The `is_match_condition<>` type trait automatically evaluates to true for functions, and for function objects with a nested `result_type` typedef. For other types the trait must be explicitly specialised, as shown above.

See Also

async_read_until(), is_match_condition, read_until(), streambuf, HTTP client example.

Custom Memory Allocation

Many asynchronous operations need to allocate an object to store state associated with the operation. For example, a Win32 implementation needs `OVERLAPPED`-derived objects to pass to Win32 API functions.

Furthermore, programs typically contain easily identifiable chains of asynchronous operations. A half duplex protocol implementation (e.g. an HTTP server) would have a single chain of operations per client (receives followed by sends). A full duplex protocol imple-

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

mentation would have two chains executing in parallel. Programs should be able to leverage this knowledge to reuse memory for all asynchronous operations in a chain.

Given a copy of a user-defined `Handler` object `h`, if the implementation needs to allocate memory associated with that handler it will execute the code:

```
void* pointer = asio_handler_allocate(size, &h);
```

Similarly, to deallocate the memory it will execute:

```
asio_handler_deallocate(pointer, size, &h);
```

These functions are located using argument-dependent lookup. The implementation provides default implementations of the above functions in the `asio` namespace:

```
void* asio_handler_allocate(size_t, ...);
void asio_handler_deallocate(void*, size_t, ...);
```

which are implemented in terms of `::operator new()` and `::operator delete()` respectively.

The implementation guarantees that the deallocation will occur before the associated handler is invoked, which means the memory is ready to be reused for any new asynchronous operations started by the handler.

The custom memory allocation functions may be called from any user-created thread that is calling a library function. The implementation guarantees that, for the asynchronous operations included the library, the implementation will not make concurrent calls to the memory allocation functions for that handler. The implementation will insert appropriate memory barriers to ensure correct memory visibility should allocation functions need to be called from different threads.

See Also

asio_handler_allocate, asio_handler_deallocate, custom memory allocation example.

Handler Tracking

To aid in debugging asynchronous programs, Boost.Asio provides support for handler tracking. When enabled by defining `BOOST_ASIO_ENABLE_HANDLER_TRACKING`, Boost.Asio writes debugging output to the standard error stream. The output records asynchronous operations and the relationships between their handlers.

This feature is useful when debugging and you need to know how your asynchronous operations are chained together, or what the pending asynchronous operations are. As an illustration, here is the output when you run the HTTP Server example, handle a single request, then shut down via Ctrl+C:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
@asio|1298160085.070638|0*1|signal_set@0x7fff50528f40.async_wait
@asio|1298160085.070888|0*2|socket@0x7fff50528f60.async_accept
@asio|1298160085.070913|0|resolver@0x7fff50528e28.cancel
@asio|1298160118.075438|>2|ec=asio.system:0
@asio|1298160118.075472|2*3|socket@0xb39048.async_receive
@asio|1298160118.075507|2*4|socket@0x7fff50528f60.async_accept
@asio|1298160118.075527|<2|
@asio|1298160118.075540|>3|ec=asio.system:0,bytes_transferred=122
@asio|1298160118.075731|3*5|socket@0xb39048.async_send
@asio|1298160118.075778|<3|
@asio|1298160118.075793|>5|ec=asio.system:0,bytes_transferred=156
@asio|1298160118.075831|5|socket@0xb39048.close
@asio|1298160118.075855|<5|
@asio|1298160122.827317|>1|ec=asio.system:0,signal_number=2
@asio|1298160122.827333|1|socket@0x7fff50528f60.close
@asio|1298160122.827359|<1|
@asio|1298160122.827370|>4|ec=asio.system:125
@asio|1298160122.827378|<4|
@asio|1298160122.827394|0|signal_set@0x7fff50528f40.cancel
```

Each line is of the form:

```
<tag>|<timestamp>|<action>|<description>
```

The `<tag>` is always `@asio`, and is used to identify and extract the handler tracking messages from the program output.

The `<timestamp>` is seconds and microseconds from 1 Jan 1970 UTC.

The `<action>` takes one of the following forms:

>n The program entered the handler number `n`. The `<description>` shows the arguments to the handler.

<n The program left handler number `n`.

!n The program left handler number n due to an exception.

~n The handler number `n` was destroyed without having been invoked. This is usually the case for any unfinished asynchronous operations when the `io_service` is destroyed.

n*m The handler number `n` created a new asynchronous operation with completion handler number `m`. The `<description>` shows what asynchronous operation was started.

n The handler number n performed some other operation. The `<description>` shows what function was called. Currently only `close()` and `cancel()` operations are logged, as these may affect the state of pending asynchronous operations.

Where the `<description>` shows a synchronous or asynchronous operation, the format is `<object-type>@<pointer>.<operation>`. For handler entry, it shows a comma-separated list of arguments and their values.

As shown above, Each handler is assigned a numeric identifier. Where the handler tracking output shows a handler number of 0, it means that the action was performed outside of any handler.

Visual Representations

The handler tracking output may be post-processed using the included `handlerviz.pl` tool to create a visual representation of the handlers (requires the GraphViz tool `dot`).

Networking

- TCP, UDP and ICMP

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- Socket Iostreams
- The BSD Socket API and Boost.Asio

TCP, UDP and ICMP

Boost.Asio provides off-the-shelf support for the internet protocols TCP, UDP and ICMP.

TCP Clients

Hostname resolution is performed using a resolver, where host and service names are looked up and converted into one or more endpoints:

```
ip::tcp::resolver resolver(my_io_service);
ip::tcp::resolver::query query("www.boost.org", "http");
ip::tcp::resolver::iterator iter = resolver.resolve(query);
ip::tcp::resolver::iterator end; // End marker.
while (iter != end)
{
  ip::tcp::endpoint endpoint = *iter++;
  std::cout << endpoint << std::endl;
}
```

The list of endpoints obtained above could contain both IPv4 and IPv6 endpoints, so a program should try each of them until it finds one that works. This keeps the client program independent of a specific IP version.

To simplify the development of protocol-independent programs, TCP clients may establish connections using the free functions connect() and async_connect(). These operations try each endpoint in a list until the socket is successfully connected. For example, a single call:

```
ip::tcp::socket socket(my_io_service);
boost::asio::connect(socket, resolver.resolve(query));
```

will synchronously try all endpoints until one is successfully connected. Similarly, an asynchronous connect may be performed by writing:

```
boost::asio::async_connect(socket_, iter,
    boost::bind(&client::handle_connect, this,
      boost::asio::placeholders::error));

// ...

void handle_connect(const error_code& error)
{
  if (!error)
  {
    // Start read or write operations.
  }
  else
  {
    // Handle error.
  }
}
```

When a specific endpoint is available, a socket can be created and connected:

```
ip::tcp::socket socket(my_io_service);
socket.connect(endpoint);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Data may be read from or written to a connected TCP socket using the receive(), async_receive(), send() or async_send() member functions. However, as these could result in short writes or reads, an application will typically use the following operations instead: read(), async_read(), write() and async_write().

TCP Servers

A program uses an acceptor to accept incoming TCP connections:

```
ip::tcp::acceptor acceptor(my_io_service, my_endpoint);
...
ip::tcp::socket socket(my_io_service);
acceptor.accept(socket);
```

After a socket has been successfully accepted, it may be read from or written to as illustrated for TCP clients above.

UDP

UDP hostname resolution is also performed using a resolver:

```
ip::udp::resolver resolver(my_io_service);
ip::udp::resolver::query query("localhost", "daytime");
ip::udp::resolver::iterator iter = resolver.resolve(query);
...
```

A UDP socket is typically bound to a local endpoint. The following code will create an IP version 4 UDP socket and bind it to the "any" address on port `12345`:

```
ip::udp::endpoint endpoint(ip::udp::v4(), 12345);
ip::udp::socket socket(my_io_service, endpoint);
```

Data may be read from or written to an unconnected UDP socket using the receive_from(), async_receive_from(), send_to() or async_send_to() member functions. For a connected UDP socket, use the receive(), async_receive(), send() or async_send() member functions.

ICMP

As with TCP and UDP, ICMP hostname resolution is performed using a resolver:

```
ip::icmp::resolver resolver(my_io_service);
ip::icmp::resolver::query query("localhost", "");
ip::icmp::resolver::iterator iter = resolver.resolve(query);
...
```

An ICMP socket may be bound to a local endpoint. The following code will create an IP version 6 ICMP socket and bind it to the "any" address:

```
ip::icmp::endpoint endpoint(ip::icmp::v6(), 0);
ip::icmp::socket socket(my_io_service, endpoint);
```

The port number is not used for ICMP.

Data may be read from or written to an unconnected ICMP socket using the receive_from(), async_receive_from(), send_to() or async_send_to() member functions.

Other Protocols

Support for other socket protocols (such as Bluetooth or IRCOMM sockets) can be added by implementing the Protocol type requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

See Also

ip::tcp, ip::udp, ip::icmp, daytime protocol tutorials, ICMP ping example.

Socket Iostreams

Boost.Asio includes classes that implement iostreams on top of sockets. These hide away the complexities associated with endpoint resolution, protocol independence, etc. To create a connection one might simply write:

```
ip::tcp::iostream stream("www.boost.org", "http");
if (!stream)
{
  // Can't connect.
}
```

The iostream class can also be used in conjunction with an acceptor to create simple servers. For example:

```
io_service ios;

ip::tcp::endpoint endpoint(tcp::v4(), 80);
ip::tcp::acceptor acceptor(ios, endpoint);

for (;;)
{
  ip::tcp::iostream stream;
  acceptor.accept(*stream.rdbuf());
  ...
}
```

Timeouts may be set by calling `expires_at()` or `expires_from_now()` to establish a deadline. Any socket operations that occur past the deadline will put the iostream into a "bad" state.

For example, a simple client program like this:

```
ip::tcp::iostream stream;
stream.expires_from_now(boost::posix_time::seconds(60));
stream.connect("www.boost.org", "http");
stream << "GET /LICENSE_1_0.txt HTTP/1.0\r\n";
stream << "Host: www.boost.org\r\n";
stream << "Accept: */*\r\n";
stream << "Connection: close\r\n\r\n";
stream.flush();
std::cout << stream.rdbuf();
```

will fail if all the socket operations combined take longer than 60 seconds.

If an error does occur, the iostream's `error()` member function may be used to retrieve the error code from the most recent system call:

```
if (!stream)
{
  std::cout << "Error: " << stream.error().message() << "\n";
}
```

See Also

ip::tcp::iostream, basic_socket_iostream, iostreams examples.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Notes

These iostream templates only support `char`, not `wchar_t`, and do not perform any code conversion.

The BSD Socket API and Boost.Asio

The Boost.Asio library includes a low-level socket interface based on the BSD socket API, which is widely implemented and supported by extensive literature. It is also used as the basis for networking APIs in other languages, like Java. This low-level interface is designed to support the development of efficient and scalable applications. For example, it permits programmers to exert finer control over the number of system calls, avoid redundant data copying, minimise the use of resources like threads, and so on.

Unsafe and error prone aspects of the BSD socket API not included. For example, the use of `int` to represent all sockets lacks type safety. The socket representation in Boost.Asio uses a distinct type for each protocol, e.g. for TCP one would use `ip::tcp::socket`, and for UDP one uses `ip::udp::socket`.

The following table shows the mapping between the BSD socket API and Boost.Asio:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

BSD Socket API Elements	Equivalents in Boost.Asio
socket descriptor - `int` (POSIX) or `SOCKET` (Windows)	For TCP: ip::tcp::socket, ip::tcp::acceptor For UDP: ip::udp::socket basic_socket, basic_stream_socket, basic_datagram_socket, basic_raw_socket
`in_addr`, `in6_addr`	ip::address, ip::address_v4, ip::address_v6
`sockaddr_in`, `sockaddr_in6`	For TCP: ip::tcp::endpoint For UDP: ip::udp::endpoint ip::basic_endpoint
`accept()`	For TCP: ip::tcp::acceptor::accept() basic_socket_acceptor::accept()
`bind()`	For TCP: ip::tcp::acceptor::bind(), ip::tcp::socket::bind() For UDP: ip::udp::socket::bind() basic_socket::bind()
`close()`	For TCP: ip::tcp::acceptor::close(), ip::tcp::socket::close() For UDP: ip::udp::socket::close() basic_socket::close()
`connect()`	For TCP: ip::tcp::socket::connect() For UDP: ip::udp::socket::connect() basic_socket::connect()
`getaddrinfo()`, `gethostbyaddr()`, `gethostbyname()`, `getnameinfo()`, `getservbyname()`, `getservbyport()`	For TCP: ip::tcp::resolver::resolve(), ip::tcp::resolver::async_resolve() For UDP: ip::udp::resolver::resolve(), ip::udp::resolver::async_resolve() ip::basic_resolver::resolve(), ip::basic_resolver::async_resolve()
`gethostname()`	ip::host_name()
`getpeername()`	For TCP: ip::tcp::socket::remote_endpoint() For UDP: ip::udp::socket::remote_endpoint() basic_socket::remote_endpoint()

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

BSD Socket API Elements	Equivalents in Boost.Asio
`getsockname()`	For TCP: ip::tcp::acceptor::local_endpoint(), ip::tcp::socket::local_endpoint() For UDP: ip::udp::socket::local_endpoint() basic_socket::local_endpoint()
`getsockopt()`	For TCP: ip::tcp::acceptor::get_option(), ip::tcp::socket::get_option() For UDP: ip::udp::socket::get_option() basic_socket::get_option()
`inet_addr()`, `inet_aton()`, `inet_pton()`	ip::address::from_string(), ip::address_v4::from_string(), ip_address_v6::from_string()
`inet_ntoa()`, `inet_ntop()`	ip::address::to_string(), ip::address_v4::to_string(), ip_address_v6::to_string()
`ioctl()`	For TCP: ip::tcp::socket::io_control() For UDP: ip::udp::socket::io_control() basic_socket::io_control()
`listen()`	For TCP: ip::tcp::acceptor::listen() basic_socket_acceptor::listen()
`poll()`, `select()`, `pselect()`	io_service::run(), io_service::run_one(), io_service::poll(), io_service::poll_one() Note: in conjunction with asynchronous operations.
`readv()`, `recv()`, `read()`	For TCP: ip::tcp::socket::read_some(), ip::tcp::socket::async_read_some(), ip::tcp::socket::receive(), ip::tcp::socket::async_receive() For UDP: ip::udp::socket::receive(), ip::udp::socket::async_receive() basic_stream_socket::read_some(), basic_stream_socket::async_read_some(), basic_stream_socket::receive(), basic_stream_socket::async_receive(), basic_datagram_socket::receive(), basic_datagram_socket::async_receive()
`recvfrom()`	For UDP: ip::udp::socket::receive_from(), ip::udp::socket::async_receive_from() basic_datagram_socket::receive_from(), basic_datagram_socket::async_receive_from()

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

BSD Socket API Elements	Equivalents in Boost.Asio
`send()`, `write()`, `writev()`	For TCP: ip::tcp::socket::write_some(), ip::tcp::socket::async_write_some(), ip::tcp::socket::send(), ip::tcp::socket::async_send() For UDP: ip::udp::socket::send(), ip::udp::socket::async_send() basic_stream_socket::write_some(), basic_stream_socket::async_write_some(), basic_stream_socket::send(), basic_stream_socket::async_send(), basic_datagram_socket::send(), basic_datagram_socket::async_send()
`sendto()`	For UDP: ip::udp::socket::send_to(), ip::udp::socket::async_send_to() basic_datagram_socket::send_to(), basic_datagram_socket::async_send_to()
`setsockopt()`	For TCP: ip::tcp::acceptor::set_option(), ip::tcp::socket::set_option() For UDP: ip::udp::socket::set_option() basic_socket::set_option()
`shutdown()`	For TCP: ip::tcp::socket::shutdown() For UDP: ip::udp::socket::shutdown() basic_socket::shutdown()
`sockatmark()`	For TCP: ip::tcp::socket::at_mark() basic_socket::at_mark()
`socket()`	For TCP: ip::tcp::acceptor::open(), ip::tcp::socket::open() For UDP: ip::udp::socket::open() basic_socket::open()
`socketpair()`	local::connect_pair() Note: POSIX operating systems only.

Timers

Long running I/O operations will often have a deadline by which they must have completed. These deadlines may be expressed as absolute times, but are often calculated relative to the current time.

As a simple example, to perform a synchronous wait operation on a timer using a relative time one may write:

```
io_service i;
...
deadline_timer t(i);
t.expires_from_now(boost::posix_time::seconds(5));
t.wait();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

More commonly, a program will perform an asynchronous wait operation on a timer:

```
void handler(boost::system::error_code ec) { ... }
...
io_service i;
...
deadline_timer t(i);
t.expires_from_now(boost::posix_time::milliseconds(400));
t.async_wait(handler);
...
i.run();
```

The deadline associated with a timer may also be obtained as a relative time:

```
boost::posix_time::time_duration time_until_expiry
  = t.expires_from_now();
```

or as an absolute time to allow composition of timers:

```
deadline_timer t2(i);
t2.expires_at(t.expires_at() + boost::posix_time::seconds(30));
```

See Also

basic_deadline_timer, deadline_timer, deadline_timer_service, timer tutorials.

Serial Ports

Boost.Asio includes classes for creating and manipulating serial ports in a portable manner. For example, a serial port may be opened using:

```
serial_port port(my_io_service, name);
```

where name is something like `"COM1"` on Windows, and `"/dev/ttyS0"` on POSIX platforms.

Once opened, the serial port may be used as a stream. This means the objects can be used with any of the read(), async_read(), write(), async_write(), read_until() or async_read_until() free functions.

The serial port implementation also includes option classes for configuring the port's baud rate, flow control type, parity, stop bits and character size.

See Also

serial_port, serial_port_base, basic_serial_port, serial_port_service, serial_port_base::baud_rate, serial_port_base::flow_control, serial_port_base::parity, serial_port_base::stop_bits, serial_port_base::character_size.

Notes

Serial ports are available on all POSIX platforms. For Windows, serial ports are only available at compile time when the I/O completion port backend is used (which is the default). A program may test for the macro `BOOST_ASIO_HAS_SERIAL_PORTS` to determine whether they are supported.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Signal Handling

Boost.Asio supports signal handling using a class called signal_set. Programs may add one or more signals to the set, and then perform an `async_wait()` operation. The specified handler will be called when one of the signals occurs. The same signal number may be registered with multiple signal_set objects, however the signal number must be used only with Boost.Asio.

```
void handler(
    const boost::system::error_code& error,
    int signal_number)
{
  if (!error)
  {
    // A signal occurred.
  }
}

...

// Construct a signal set registered for process termination.
boost::asio::signal_set signals(io_service, SIGINT, SIGTERM);

// Start an asynchronous wait for one of the signals to occur.
signals.async_wait(handler);
```

Signal handling also works on Windows, as the Microsoft Visual C++ runtime library maps console events like Ctrl+C to the equivalent signal.

See Also

signal_set, HTTP server example.

POSIX-Specific Functionality

UNIX Domain Sockets

Stream-Oriented File Descriptors

Fork

UNIX Domain Sockets

Boost.Asio provides basic support UNIX domain sockets (also known as local sockets). The simplest use involves creating a pair of connected sockets. The following code:

```
local::stream_protocol::socket socket1(my_io_service);
local::stream_protocol::socket socket2(my_io_service);
local::connect_pair(socket1, socket2);
```

will create a pair of stream-oriented sockets. To do the same for datagram-oriented sockets, use:

```
local::datagram_protocol::socket socket1(my_io_service);
local::datagram_protocol::socket socket2(my_io_service);
local::connect_pair(socket1, socket2);
```

A UNIX domain socket server may be created by binding an acceptor to an endpoint, in much the same way as one does for a TCP server:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
::unlink("/tmp/foobar"); // Remove previous binding.
local::stream_protocol::endpoint ep("/tmp/foobar");
local::stream_protocol::acceptor acceptor(my_io_service, ep);
local::stream_protocol::socket socket(my_io_service);
acceptor.accept(socket);
```

A client that connects to this server might look like:

```
local::stream_protocol::endpoint ep("/tmp/foobar");
local::stream_protocol::socket socket(my_io_service);
socket.connect(ep);
```

Transmission of file descriptors or credentials across UNIX domain sockets is not directly supported within Boost.Asio, but may be achieved by accessing the socket's underlying descriptor using the native() member function.

See Also

local::connect_pair, local::datagram_protocol, local::datagram_protocol::endpoint, local::datagram_protocol::socket, local::stream_protocol, local::stream_protocol::acceptor, local::stream_protocol::endpoint, local::stream_protocol::iostream, local::stream_protocol::socket, UNIX domain sockets examples.

Notes

UNIX domain sockets are only available at compile time if supported by the target operating system. A program may test for the macro `BOOST_ASIO_HAS_LOCAL_SOCKETS` to determine whether they are supported.

Stream-Oriented File Descriptors

Boost.Asio includes classes added to permit synchronous and asynchronous read and write operations to be performed on POSIX file descriptors, such as pipes, standard input and output, and various devices (but *not* regular files).

For example, to perform read and write operations on standard input and output, the following objects may be created:

```
posix::stream_descriptor in(my_io_service, ::dup(STDIN_FILENO));
posix::stream_descriptor out(my_io_service, ::dup(STDOUT_FILENO));
```

These are then used as synchronous or asynchronous read and write streams. This means the objects can be used with any of the read(), async_read(), write(), async_write(), read_until() or async_read_until() free functions.

See Also

posix::stream_descriptor, posix::basic_stream_descriptor, posix::stream_descriptor_service, Chat example.

Notes

POSIX stream descriptors are only available at compile time if supported by the target operating system. A program may test for the macro `BOOST_ASIO_HAS_POSIX_STREAM_DESCRIPTOR` to determine whether they are supported.

Fork

Boost.Asio supports programs that utilise the `fork()` system call. Provided the program calls `io_service.notify_fork()` at the appropriate times, Boost.Asio will recreate any internal file descriptors (such as the "self-pipe trick" descriptor used for waking up a reactor). The notification is usually performed as follows:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
io_service_.notify_fork(boost::asio::io_service::fork_prepare);
if (fork() == 0)
{
  io_service_.notify_fork(boost::asio::io_service::fork_child);
  ...
}
else
{
  io_service_.notify_fork(boost::asio::io_service::fork_parent);
  ...
}
```

User-defined services can also be made fork-aware by overriding the `io_service::service::fork_service()` virtual function.

Note that any file descriptors accessible via Boost.Asio's public API (e.g. the descriptors underlying `basic_socket<>`, `posix::stream_descriptor`, etc.) are not altered during a fork. It is the program's responsibility to manage these as required.

See Also

io_service::notify_fork(), io_service::fork_event, io_service::service::fork_service(), Fork examples.

Windows-Specific Functionality

Stream-Oriented HANDLEs

Random-Access HANDLEs

Object HANDLEs

Stream-Oriented HANDLEs

Boost.Asio contains classes to allow asynchronous read and write operations to be performed on Windows `HANDLE`s, such as named pipes.

For example, to perform asynchronous operations on a named pipe, the following object may be created:

```
HANDLE handle = ::CreateFile(...);
windows::stream_handle pipe(my_io_service, handle);
```

These are then used as synchronous or asynchronous read and write streams. This means the objects can be used with any of the read(), async_read(), write(), async_write(), read_until() or async_read_until() free functions.

The kernel object referred to by the `HANDLE` must support use with I/O completion ports (which means that named pipes are supported, but anonymous pipes and console streams are not).

See Also

windows::stream_handle, windows::basic_stream_handle, windows::stream_handle_service.

Notes

Windows stream `HANDLE`s are only available at compile time when targeting Windows and only when the I/O completion port backend is used (which is the default). A program may test for the macro `BOOST_ASIO_HAS_WINDOWS_STREAM_HANDLE` to determine whether they are supported.

Random-Access HANDLEs

Boost.Asio provides Windows-specific classes that permit asynchronous read and write operations to be performed on HANDLEs that refer to regular files.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

For example, to perform asynchronous operations on a file the following object may be created:

```
HANDLE handle = ::CreateFile(...);
windows::random_access_handle file(my_io_service, handle);
```

Data may be read from or written to the handle using one of the `read_some_at()`, `async_read_some_at()`, `write_some_at()` or `async_write_some_at()` member functions. However, like the equivalent functions (`read_some()`, etc.) on streams, these functions are only required to transfer one or more bytes in a single operation. Therefore free functions called read_at(), async_read_at(), write_at() and async_write_at() have been created to repeatedly call the corresponding `*_some_at()` function until all data has been transferred.

See Also

windows::random_access_handle, windows::basic_random_access_handle, windows::random_access_handle_service.

Notes

Windows random-access `HANDLE`s are only available at compile time when targeting Windows and only when the I/O completion port backend is used (which is the default). A program may test for the macro `BOOST_ASIO_HAS_WINDOWS_RANDOM_ACCESS_HANDLE` to determine whether they are supported.

Object HANDLEs

Boost.Asio provides Windows-specific classes that permit asynchronous wait operations to be performed on HANDLEs to kernel objects of the following types:

- Change notification
- Console input
- Event
- Memory resource notification
- Process
- Semaphore
- Thread
- Waitable timer

For example, to perform asynchronous operations on an event, the following object may be created:

```
HANDLE handle = ::CreateEvent(...);
windows::object_handle file(my_io_service, handle);
```

The `wait()` and `async_wait()` member functions may then be used to wait until the kernel object is signalled.

See Also

windows::object_handle, windows::basic_object_handle, windows::object_handle_service.

Notes

Windows object `HANDLE`s are only available at compile time when targeting Windows. Programs may test for the macro `BOOST_ASIO_HAS_WINDOWS_OBJECT_HANDLE` to determine whether they are supported.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

SSL

Boost.Asio contains classes and class templates for basic SSL support. These classes allow encrypted communication to be layered on top of an existing stream, such as a TCP socket.

Before creating an encrypted stream, an application must construct an SSL context object. This object is used to set SSL options such as verification mode, certificate files, and so on. As an illustration, client-side initialisation may look something like:

```
ssl::context ctx(ssl::context::sslv23);
ctx.set_verify_mode(ssl::verify_peer);
ctx.load_verify_file("ca.pem");
```

To use SSL with a TCP socket, one may write:

```
ssl::stream<ip::tcp::socket> ssl_sock(my_io_service, ctx);
```

To perform socket-specific operations, such as establishing an outbound connection or accepting an incoming one, the underlying socket must first be obtained using the `ssl::stream` template's `lowest_layer()` member function:

```
ip::tcp::socket::lowest_layer_type& sock = ssl_sock.lowest_layer();
sock.connect(my_endpoint);
```

In some use cases the underlying stream object will need to have a longer lifetime than the SSL stream, in which case the template parameter should be a reference to the stream type:

```
ip::tcp::socket sock(my_io_service);
ssl::stream<ip::tcp::socket&> ssl_sock(sock, ctx);
```

SSL handshaking must be performed prior to transmitting or receiving data over an encrypted connection. This is accomplished using the `ssl::stream` template's handshake() or async_handshake() member functions.

Once connected, SSL stream objects are used as synchronous or asynchronous read and write streams. This means the objects can be used with any of the read(), async_read(), write(), async_write(), read_until() or async_read_until() free functions.

Certificate Verification

Boost.Asio provides various methods for configuring the way SSL certificates are verified:

- ssl::context::set_default_verify_paths()
- ssl::context::set_verify_mode()
- ssl::context::set_verify_callback()
- ssl::context::load_verify_file()
- ssl::stream::set_verify_mode()
- ssl::stream::set_verify_callback()

To simplify use cases where certificates are verified according to the rules in RFC 2818 (certificate verification for HTTPS), Boost.Asio provides a reusable verification callback as a function object:

- ssl::rfc2818_verification

The following example shows verification of a remote host's certificate according to the rules used by HTTPS:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
using boost::asio::ip::tcp;
namespace ssl = boost::asio::ssl;
typedef ssl::stream<tcp::socket> ssl_socket;

// Create a context that uses the default paths for
// finding CA certificates.
ssl::context ctx(ssl::context::sslv23);
ctx.set_default_verify_paths();

// Open a socket and connect it to the remote host.
boost::asio::io_service io_service;
ssl_socket sock(io_service, ctx);
tcp::resolver resolver(io_service);
tcp::resolver::query query("host.name", "https");
boost::asio::connect(sock.lowest_layer(), resolver.resolve(query));
sock.lowest_layer().set_option(tcp::no_delay(true));

// Perform SSL handshake and verify the remote host's
// certificate.
sock.set_verify_mode(ssl::verify_peer);
sock.set_verify_callback(ssl::rfc2818_verification("host.name"));
sock.handshake(ssl_socket::client);

// ... read and write as normal ...
```

SSL and Threads

SSL stream objects perform no locking of their own. Therefore, it is essential that all asynchronous SSL operations are performed in an implicit or explicit strand. Note that this means that no synchronisation is required (and so no locking overhead is incurred) in single threaded programs.

See Also

ssl::context, ssl::rfc2818_verification, ssl::stream, SSL example.

Notes

OpenSSL is required to make use of Boost.Asio's SSL support. When an application needs to use OpenSSL functionality that is not wrapped by Boost.Asio, the underlying OpenSSL types may be obtained by calling `ssl::context::native_handle()` or `ssl::stream::native_handle()`.

C++ 2011 Support

Movable I/O Objects

Movable Handlers

Variadic Templates

Array Container

Atomics

Shared Pointers

Chrono

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Movable I/O Objects

When move support is available (via rvalue references), Boost.Asio allows move construction and assignment of sockets, serial ports, POSIX descriptors and Windows handles.

Move support allows you to write code like:

```
tcp::socket make_socket(io_service& i)
{
  tcp::socket s(i);
  ...
  std::move(s);
}
```

or:

```
class connection : public enable_shared_from_this<connection>
{
private:
  tcp::socket socket_;
  ...
public:
  connection(tcp::socket&& s) : socket_(std::move(s)) {}
  ...
};

...

class server
{
private:
  tcp::acceptor acceptor_;
  tcp::socket socket_;
  ...
  void handle_accept(error_code ec)
  {
    if (!ec)
      std::make_shared<connection>(std::move(socket_))->go();
    acceptor_.async_accept(socket_, ...);
  }
  ...
};
```

as well as:

```
std::vector<tcp::socket> sockets;
sockets.push_back(tcp::socket(...));
```

A word of warning: There is nothing stopping you from moving these objects while there are pending asynchronous operations, but it is unlikely to be a good idea to do so. In particular, composed operations like async_read() store a reference to the stream object. Moving during the composed operation means that the composed operation may attempt to access a moved-from object.

Move support is automatically enabled for `g++` 4.5 and later, when the `-std=c++0x` or `-std=gnu++0x` compiler options are used. It may be disabled by defining `BOOST_ASIO_DISABLE_MOVE`, or explicitly enabled for other compilers by defining `BOOST_ASIO_HAS_MOVE`. Note that these macros also affect the availability of movable handlers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Movable Handlers

As an optimisation, user-defined completion handlers may provide move constructors, and Boost.Asio's implementation will use a handler's move constructor in preference to its copy constructor. In certain circumstances, Boost.Asio may be able to eliminate all calls to a handler's copy constructor. However, handler types are still required to be copy constructible.

When move support is enabled, asynchronous that are documented as follows:

```
template <typename Handler>
void async_XYZ(..., Handler handler);
```

are actually declared as:

```
template <typename Handler>
void async_XYZ(..., Handler&& handler);
```

The handler argument is perfectly forwarded and the move construction occurs within the body of `async_XYZ()`. This ensures that all other function arguments are evaluated prior to the move. This is critical when the other arguments to `async_XYZ()` are members of the handler. For example:

```
struct my_operation
{
  shared_ptr<tcp::socket> socket;
  shared_ptr<vector<char>> buffer;
  ...
  void operator(error_code ec, size_t length)
  {
    ...
    socket->async_read_some(boost::asio::buffer(*buffer), std::move(*this));
    ...
  }
};
```

Move support is automatically enabled for `g++` 4.5 and later, when the `-std=c++0x` or `-std=gnu++0x` compiler options are used. It may be disabled by defining `BOOST_ASIO_DISABLE_MOVE`, or explicitly enabled for other compilers by defining `BOOST_ASIO_HAS_MOVE`. Note that these macros also affect the availability of movable I/O objects.

Variadic Templates

When supported by a compiler, Boost.Asio can use variadic templates to implement the basic_socket_streambuf::connect() and basic_socket_iostream::connect() functions.

Support for variadic templates is automatically enabled for `g++` 4.3 and later, when the `-std=c++0x` or `-std=gnu++0x` compiler options are used. It may be disabled by defining `BOOST_ASIO_DISABLE_VARIADIC_TEMPLATES`, or explicitly enabled for other compilers by defining `BOOST_ASIO_HAS_VARIADIC_TEMPLATES`.

Array Container

Where the standard library provides `std::array<>`, Boost.Asio:

- Provides overloads for the buffer() function.
- Uses it in preference to `boost::array<>` for the ip::address_v4::bytes_type and ip::address_v6::bytes_type types.
- Uses it in preference to `boost::array<>` where a fixed size array type is needed in the implementation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Support for `std::array<>` is automatically enabled for `g++` 4.3 and later, when the `-std=c++0x` or `-std=gnu++0x` compiler options are used, as well as for Microsoft Visual C++ 10. It may be disabled by defining `BOOST_ASIO_DISABLE_STD_ARRAY`, or explicitly enabled for other compilers by defining `BOOST_ASIO_HAS_STD_ARRAY`.

Atomics

Boost.Asio's implementation can use `std::atomic<>` in preference to `boost::detail::atomic_count`.

Support for the standard atomic integer template is automatically enabled for `g++` 4.5 and later, when the `-std=c++0x` or `-std=gnu++0x` compiler options are used. It may be disabled by defining `BOOST_ASIO_DISABLE_STD_ATOMIC`, or explicitly enabled for other compilers by defining `BOOST_ASIO_HAS_STD_ATOMIC`.

Shared Pointers

Boost.Asio's implementation can use `std::shared_ptr<>` and `std::weak_ptr<>` in preference to the Boost equivalents.

Support for the standard smart pointers is automatically enabled for `g++` 4.3 and later, when the `-std=c++0x` or `-std=gnu++0x` compiler options are used, as well as for Microsoft Visual C++ 10. It may be disabled by defining `BOOST_ASIO_DISABLE_STD_SHARED_PTR`, or explicitly enabled for other compilers by defining `BOOST_ASIO_HAS_STD_SHARED_PTR`.

Chrono

Boost.Asio provides timers based on the `std::chrono` facilities via the basic_waitable_timer class template. The typedefs system_timer, steady_timer and high_resolution_timer utilise the standard clocks `system_clock`, `steady_clock` and `high_resolution_clock` respectively.

Support for the `std::chrono` facilities is automatically enabled for `g++` 4.6 and later, when the `-std=c++0x` or `-std=gnu++0x` compiler options are used. (Note that, for `g++`, the draft-standard `monotonic_clock` is used in place of `steady_clock`.) Support may be disabled by defining `BOOST_ASIO_DISABLE_STD_CHRONO`, or explicitly enabled for other compilers by defining `BOOST_ASIO_HAS_STD_CHRONO`.

When standard `chrono` is unavailable, Boost.Asio will otherwise use the Boost.Chrono library. The basic_waitable_timer class template may be used with either.

Platform-Specific Implementation Notes

This section lists platform-specific implementation details, such as the default demultiplexing mechanism, the number of threads created internally, and when threads are created.

Linux Kernel 2.4

Demultiplexing mechanism:

- Uses `select` for demultiplexing. This means that the number of file descriptors in the process cannot be permitted to exceed `FD_SETSIZE`.

Threads:

- Demultiplexing using `select` is performed in one of the threads that calls `io_service::run()`, `io_service::run_one()`, `io_service::poll()` or `io_service::poll_one()`.
- An additional thread per `io_service` is used to emulate asynchronous host resolution. This thread is created on the first call to either `ip::tcp::resolver::async_resolve()` or `ip::udp::resolver::async_resolve()`.

Scatter-Gather:

- At most `min(64,IOV_MAX)` buffers may be transferred in a single operation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Linux Kernel 2.6

Demultiplexing mechanism:

- Uses `epoll` for demultiplexing.

Threads:

- Demultiplexing using `epoll` is performed in one of the threads that calls `io_service::run()`, `io_service::run_one()`, `io_service::poll()` or `io_service::poll_one()`.
- An additional thread per `io_service` is used to emulate asynchronous host resolution. This thread is created on the first call to either `ip::tcp::resolver::async_resolve()` or `ip::udp::resolver::async_resolve()`.

Scatter-Gather:

- At most `min(64,IOV_MAX)` buffers may be transferred in a single operation.

Solaris

Demultiplexing mechanism:

- Uses `/dev/poll` for demultiplexing.

Threads:

- Demultiplexing using `/dev/poll` is performed in one of the threads that calls `io_service::run()`, `io_service::run_one()`, `io_service::poll()` or `io_service::poll_one()`.
- An additional thread per `io_service` is used to emulate asynchronous host resolution. This thread is created on the first call to either `ip::tcp::resolver::async_resolve()` or `ip::udp::resolver::async_resolve()`.

Scatter-Gather:

- At most `min(64,IOV_MAX)` buffers may be transferred in a single operation.

QNX Neutrino

Demultiplexing mechanism:

- Uses `select` for demultiplexing. This means that the number of file descriptors in the process cannot be permitted to exceed `FD_SETSIZE`.

Threads:

- Demultiplexing using `select` is performed in one of the threads that calls `io_service::run()`, `io_service::run_one()`, `io_service::poll()` or `io_service::poll_one()`.
- An additional thread per `io_service` is used to emulate asynchronous host resolution. This thread is created on the first call to either `ip::tcp::resolver::async_resolve()` or `ip::udp::resolver::async_resolve()`.

Scatter-Gather:

- At most `min(64,IOV_MAX)` buffers may be transferred in a single operation.

Mac OS X

Demultiplexing mechanism:

- Uses `kqueue` for demultiplexing.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Threads:

- Demultiplexing using `kqueue` is performed in one of the threads that calls `io_service::run()`, `io_service::run_one()`, `io_service::poll()` or `io_service::poll_one()`.
- An additional thread per `io_service` is used to emulate asynchronous host resolution. This thread is created on the first call to either `ip::tcp::resolver::async_resolve()` or `ip::udp::resolver::async_resolve()`.

Scatter-Gather:

- At most `min(64,IOV_MAX)` buffers may be transferred in a single operation.

FreeBSD

Demultiplexing mechanism:

- Uses `kqueue` for demultiplexing.

Threads:

- Demultiplexing using `kqueue` is performed in one of the threads that calls `io_service::run()`, `io_service::run_one()`, `io_service::poll()` or `io_service::poll_one()`.
- An additional thread per `io_service` is used to emulate asynchronous host resolution. This thread is created on the first call to either `ip::tcp::resolver::async_resolve()` or `ip::udp::resolver::async_resolve()`.

Scatter-Gather:

- At most `min(64,IOV_MAX)` buffers may be transferred in a single operation.

AIX

Demultiplexing mechanism:

- Uses `select` for demultiplexing. This means that the number of file descriptors in the process cannot be permitted to exceed `FD_SETSIZE`.

Threads:

- Demultiplexing using `select` is performed in one of the threads that calls `io_service::run()`, `io_service::run_one()`, `io_service::poll()` or `io_service::poll_one()`.
- An additional thread per `io_service` is used to emulate asynchronous host resolution. This thread is created on the first call to either `ip::tcp::resolver::async_resolve()` or `ip::udp::resolver::async_resolve()`.

Scatter-Gather:

- At most `min(64,IOV_MAX)` buffers may be transferred in a single operation.

HP-UX

Demultiplexing mechanism:

- Uses `select` for demultiplexing. This means that the number of file descriptors in the process cannot be permitted to exceed `FD_SETSIZE`.

Threads:

- Demultiplexing using `select` is performed in one of the threads that calls `io_service::run()`, `io_service::run_one()`, `io_service::poll()` or `io_service::poll_one()`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- An additional thread per `io_service` is used to emulate asynchronous host resolution. This thread is created on the first call to either `ip::tcp::resolver::async_resolve()` or `ip::udp::resolver::async_resolve()`.

Scatter-Gather:

- At most `min(64,IOV_MAX)` buffers may be transferred in a single operation.

Tru64

Demultiplexing mechanism:

- Uses `select` for demultiplexing. This means that the number of file descriptors in the process cannot be permitted to exceed `FD_SETSIZE`.

Threads:

- Demultiplexing using `select` is performed in one of the threads that calls `io_service::run()`, `io_service::run_one()`, `io_service::poll()` or `io_service::poll_one()`.
- An additional thread per `io_service` is used to emulate asynchronous host resolution. This thread is created on the first call to either `ip::tcp::resolver::async_resolve()` or `ip::udp::resolver::async_resolve()`.

Scatter-Gather:

- At most `min(64,IOV_MAX)` buffers may be transferred in a single operation.

Windows 95, 98 and Me

Demultiplexing mechanism:

- Uses `select` for demultiplexing.

Threads:

- Demultiplexing using `select` is performed in one of the threads that calls `io_service::run()`, `io_service::run_one()`, `io_service::poll()` or `io_service::poll_one()`.
- An additional thread per `io_service` is used to emulate asynchronous host resolution. This thread is created on the first call to either `ip::tcp::resolver::async_resolve()` or `ip::udp::resolver::async_resolve()`.

Scatter-Gather:

- For sockets, at most 16 buffers may be transferred in a single operation.

Windows NT, 2000, XP, 2003 and Vista

Demultiplexing mechanism:

- Uses overlapped I/O and I/O completion ports for all asynchronous socket operations except for asynchronous connect.
- Uses `select` for emulating asynchronous connect.

Threads:

- Demultiplexing using I/O completion ports is performed in all threads that call `io_service::run()`, `io_service::run_one()`, `io_service::poll()` or `io_service::poll_one()`.
- An additional thread per `io_service` is used to trigger timers. This thread is created on construction of the first `deadline_timer` or `deadline_timer_service` objects.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- An additional thread per `io_service` is used for the `select` demultiplexing. This thread is created on the first call to `async_connect()`.
- An additional thread per `io_service` is used to emulate asynchronous host resolution. This thread is created on the first call to either `ip::tcp::resolver::async_resolve()` or `ip::udp::resolver::async_resolve()`.

Scatter-Gather:

- For sockets, at most 64 buffers may be transferred in a single operation.
- For stream-oriented handles, only one buffer may be transferred in a single operation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Using Boost.Asio

Supported Platforms

The following platforms and compilers have been tested:

- Win32 and Win64 using Visual C++ 7.1 and Visual C++ 8.0.
- Win32 using MinGW.
- Win32 using Cygwin. (`__USE_W32_SOCKETS` must be defined.)
- Linux (2.4 or 2.6 kernels) using g++ 3.3 or later.
- Solaris using g++ 3.3 or later.
- Mac OS X 10.4 using g++ 3.3 or later.

The following platforms may also work:

- AIX 5.3 using XL C/C++ v9.
- HP-UX 11i v3 using patched aC++ A.06.14.
- QNX Neutrino 6.3 using g++ 3.3 or later.
- Solaris using Sun Studio 11 or later.
- Tru64 v5.1 using Compaq C++ v7.1.
- Win32 using Borland C++ 5.9.2

Dependencies

The following libraries must be available in order to link programs that use Boost.Asio:

- Boost.System for the `boost::system::error_code` and `boost::system::system_error` classes.
- Boost.Regex (optional) if you use any of the `read_until()` or `async_read_until()` overloads that take a `boost::regex` parameter.
- OpenSSL (optional) if you use Boost.Asio's SSL support.

Furthermore, some of the examples also require the Boost.Thread, Boost.Date_Time or Boost.Serialization libraries.

Note

With MSVC or Borland C++ you may want to add `-DBOOST_DATE_TIME_NO_LIB` and `-DBOOST_REGEX_NO_LIB` to your project settings to disable autolinking of the Boost.Date_Time and Boost.Regex libraries respectively. Alternatively, you may choose to build these libraries and link to them.

Building Boost Libraries

You may build the subset of Boost libraries required to use Boost.Asio and its examples by running the following command from the root of the Boost download package:

```
bjam --with-system --with-thread --with-date_time --with-regex --with-serialization stage
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This assumes that you have already built `bjam`. Consult the Boost.Build documentation for more details.

Optional separate compilation

By default, Boost.Asio is a header-only library. However, some developers may prefer to build Boost.Asio using separately compiled source code. To do this, add `#include <boost/asio/impl/src.hpp>` to one (and only one) source file in a program, then build the program with `BOOST_ASIO_SEPARATE_COMPILATION` defined in the project/compiler settings. Alternatively, `BOOST_ASIO_DYN_LINK` may be defined to build a separately-compiled Boost.Asio as part of a shared library.

If using Boost.Asio's SSL support, you will also need to add `#include <boost/asio/ssl/impl/src.hpp>`.

Macros

The macros listed in the table below may be used to control the behaviour of Boost.Asio.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Macro	Description
`BOOST_ASIO_ENABLE_BUFFER_DEBUGGING`	Enables Boost.Asio's buffer debugging support, which can help identify when invalid buffers are used in read or write operations (e.g. if a std::string object being written is destroyed before the write operation completes). When using Microsoft Visual C++, this macro is defined automatically if the compiler's iterator debugging support is enabled, unless `BOOST_ASIO_DISABLE_BUFFER_DEBUGGING` has been defined. When using g++, this macro is defined automatically if standard library debugging is enabled (`_GLIBCXX_DEBUG` is defined), unless `BOOST_ASIO_DISABLE_BUFFER_DEBUGGING` has been defined.
`BOOST_ASIO_DISABLE_BUFFER_DEBUGGING`	Explictly disables Boost.Asio's buffer debugging support.
`BOOST_ASIO_DISABLE_DEV_POLL`	Explicitly disables `/dev/poll` support on Solaris, forcing the use of a `select`-based implementation.
`BOOST_ASIO_DISABLE_EPOLL`	Explicitly disables `epoll` support on Linux, forcing the use of a `select`-based implementation.
`BOOST_ASIO_DISABLE_EVENTFD`	Explicitly disables `eventfd` support on Linux, forcing the use of a pipe to interrupt blocked epoll/select system calls.
`BOOST_ASIO_DISABLE_KQUEUE`	Explicitly disables `kqueue` support on Mac OS X and BSD variants, forcing the use of a `select`-based implementation.
`BOOST_ASIO_DISABLE_IOCP`	Explicitly disables I/O completion ports support on Windows, forcing the use of a `select`-based implementation.
`BOOST_ASIO_DISABLE_THREADS`	Explicitly disables Boost.Asio's threading support, independent of whether or not Boost as a whole supports threads.
`BOOST_ASIO_NO_WIN32_LEAN_AND_MEAN`	By default, Boost.Asio will automatically define `WIN32_LEAN_AND_MEAN` when compiling for Windows, to minimise the number of Windows SDK header files and features that are included. The presence of `BOOST_ASIO_NO_WIN32_LEAN_AND_MEAN` prevents `WIN32_LEAN_AND_MEAN` from being defined.
`BOOST_ASIO_NO_NOMINMAX`	By default, Boost.Asio will automatically define `NOMINMAX` when compiling for Windows, to suppress the definition of the `min()` and `max()` macros. The presence of `BOOST_ASIO_NO_NOMINMAX` prevents `NOMINMAX` from being defined.
`BOOST_ASIO_NO_DEFAULT_LINKED_LIBS`	When compiling for Windows using Microsoft Visual C++ or Borland C++, Boost.Asio will automatically link in the necessary Windows SDK libraries for sockets support (i.e. `ws2_32.lib` and `mswsock.lib`, or `ws2.lib` when building for Windows CE). The `BOOST_ASIO_NO_DEFAULT_LINKED_LIBS` macro prevents these libraries from being linked.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Macro	Description
`BOOST_ASIO_SOCKET_STREAMBUF_MAX_ARITY`	Determines the maximum number of arguments that may be passed to the `basic_socket_streambuf` class template's `connect` member function. Defaults to 5.
`BOOST_ASIO_SOCKET_IOSTREAM_MAX_ARITY`	Determines the maximum number of arguments that may be passed to the `basic_socket_iostream` class template's constructor and `connect` member function. Defaults to 5.
`BOOST_ASIO_ENABLE_CANCELIO`	Enables use of the `CancelIo` function on older versions of Windows. If not enabled, calls to `cancel()` on a socket object will always fail with `asio::error::operation_not_supported` when run on Windows XP, Windows Server 2003, and earlier versions of Windows. When running on Windows Vista, Windows Server 2008, and later, the `CancelIoEx` function is always used. The `CancelIo` function has two issues that should be considered before enabling its use: * It will only cancel asynchronous operations that were initiated in the current thread. * It can appear to complete without error, but the request to cancel the unfinished operations may be silently ignored by the operating system. Whether it works or not seems to depend on the drivers that are installed. For portable cancellation, consider using one of the following alternatives: * Disable asio's I/O completion port backend by defining BOOST_ASIO_DISABLE_IOCP. * Use the socket object's close() function to simultaneously cancel the outstanding operations and close the socket.
`BOOST_ASIO_NO_TYPEID`	Disables uses of the `typeid` operator in Boost.Asio. Defined automatically if `BOOST_NO_TYPEID` is defined.
`BOOST_ASIO_HASH_MAP_BUCKETS`	Determines the number of buckets in Boost.Asio's internal `hash_map` objects. The value should be a comma separated list of prime numbers, in ascending order. The `hash_map` implementation will automatically increase the number of buckets as the number of elements in the map increases. Some examples: * Defining `BOOST_ASIO_HASH_MAP_BUCKETS` to `1021` means that the `hash_map` objects will always contain 1021 buckets, irrespective of the number of elements in the map. * Defining `BOOST_ASIO_HASH_MAP_BUCKETS` to `53,389,1543` means that the `hash_map` objects will initially contain 53 buckets. The number of buckets will be increased to 389 and then 1543 as elements are added to the map.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Mailing List

A mailing list specifically for Boost.Asio may be found on SourceForge.net. Newsgroup access is provided via Gmane.

Wiki

Users are encouraged to share examples, tips and FAQs on the Boost.Asio wiki, which is located at http://think-async.com/Asio/.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Tutorial

Basic Skills

The tutorial programs in this first section introduce the fundamental concepts required to use the asio toolkit. Before plunging into the complex world of network programming, these tutorial programs illustrate the basic skills using simple asynchronous timers.

- Timer.1 - Using a timer synchronously
- Timer.2 - Using a timer asynchronously
- Timer.3 - Binding arguments to a handler
- Timer.4 - Using a member function as a handler
- Timer.5 - Synchronising handlers in multithreaded programs

Introduction to Sockets

The tutorial programs in this section show how to use asio to develop simple client and server programs. These tutorial programs are based around the daytime protocol, which supports both TCP and UDP.

The first three tutorial programs implement the daytime protocol using TCP.

- Daytime.1 - A synchronous TCP daytime client
- Daytime.2 - A synchronous TCP daytime server
- Daytime.3 - An asynchronous TCP daytime server

The next three tutorial programs implement the daytime protocol using UDP.

- Daytime.4 - A synchronous UDP daytime client
- Daytime.5 - A synchronous UDP daytime server
- Daytime.6 - An asynchronous UDP daytime server

The last tutorial program in this section demonstrates how asio allows the TCP and UDP servers to be easily combined into a single program.

- Daytime.7 - A combined TCP/UDP asynchronous server

Timer.1 - Using a timer synchronously

This tutorial program introduces asio by showing how to perform a blocking wait on a timer.

We start by including the necessary header files.

All of the asio classes can be used by simply including the `"asio.hpp"` header file.

```
#include <iostream>
#include <boost/asio.hpp>
```

Since this example uses timers, we need to include the appropriate Boost.Date_Time header file for manipulating times.

```
#include <boost/date_time/posix_time/posix_time.hpp>
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

All programs that use asio need to have at least one io_service object. This class provides access to I/O functionality. We declare an object of this type first thing in the main function.

```
int main()
{
  boost::asio::io_service io;
```

Next we declare an object of type boost::asio::deadline_timer. The core asio classes that provide I/O functionality (or as in this case timer functionality) always take a reference to an io_service as their first constructor argument. The second argument to the constructor sets the timer to expire 5 seconds from now.

```
  boost::asio::deadline_timer t(io, boost::posix_time::seconds(5));
```

In this simple example we perform a blocking wait on the timer. That is, the call to deadline_timer::wait() will not return until the timer has expired, 5 seconds after it was created (i.e. not from when the wait starts).

A deadline timer is always in one of two states: "expired" or "not expired". If the deadline_timer::wait() function is called on an expired timer, it will return immediately.

```
  t.wait();
```

Finally we print the obligatory `"Hello, world!"` message to show when the timer has expired.

```
  std::cout << "Hello, world!\n";

  return 0;
}
```

See the full source listing

Return to the tutorial index

Next: Timer.2 - Using a timer asynchronously

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Source listing for Timer.1

```
//
// timer.cpp
// ~~~~~~~~~
//
// Copyright (c) 2003-2012 Christopher M. Kohlhoff (chris at kohlhoff dot com)
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//

#include <iostream>
#include <boost/asio.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>

int main()
{
  boost::asio::io_service io;

  boost::asio::deadline_timer t(io, boost::posix_time::seconds(5));
  t.wait();

  std::cout << "Hello, world!\n";

  return 0;
}
```

Return to Timer.1 - Using a timer synchronously

Timer.2 - Using a timer asynchronously

This tutorial program demonstrates how to use asio's asynchronous callback functionality by modifying the program from tutorial Timer.1 to perform an asynchronous wait on the timer.

```
#include <iostream>
#include <boost/asio.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>
```

Using asio's asynchronous functionality means having a callback function that will be called when an asynchronous operation completes. In this program we define a function called `print` to be called when the asynchronous wait finishes.

```
void print(const boost::system::error_code& /*e*/)
{
  std::cout << "Hello, world!\n";
}

int main()
{
  boost::asio::io_service io;

  boost::asio::deadline_timer t(io, boost::posix_time::seconds(5));
```

Next, instead of doing a blocking wait as in tutorial Timer.1, we call the deadline_timer::async_wait() function to perform an asynchronous wait. When calling this function we pass the `print` callback handler that was defined above.

```
  t.async_wait(&print);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Finally, we must call the io_service::run() member function on the io_service object.

The asio library provides a guarantee that callback handlers will only be called from threads that are currently calling io_service::run(). Therefore unless the io_service::run() function is called the callback for the asynchronous wait completion will never be invoked.

The io_service::run() function will also continue to run while there is still "work" to do. In this example, the work is the asynchronous wait on the timer, so the call will not return until the timer has expired and the callback has completed.

It is important to remember to give the io_service some work to do before calling io_service::run(). For example, if we had omitted the above call to deadline_timer::async_wait(), the io_service would not have had any work to do, and consequently io_service::run() would have returned immediately.

```
  io.run();

  return 0;
}
```

See the full source listing

Return to the tutorial index

Previous: Timer.1 - Using a timer synchronously

Next: Timer.3 - Binding arguments to a handler

Source listing for Timer.2

```
//
// timer.cpp
// ~~~~~~~~~
//
// Copyright (c) 2003-2012 Christopher M. Kohlhoff (chris at kohlhoff dot com)
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//

#include <iostream>
#include <boost/asio.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>

void print(const boost::system::error_code& /*e*/)
{
  std::cout << "Hello, world!\n";
}

int main()
{
  boost::asio::io_service io;

  boost::asio::deadline_timer t(io, boost::posix_time::seconds(5));
  t.async_wait(&print);

  io.run();

  return 0;
}
```

Return to Timer.2 - Using a timer asynchronously

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Timer.3 - Binding arguments to a handler

In this tutorial we will modify the program from tutorial Timer.2 so that the timer fires once a second. This will show how to pass additional parameters to your handler function.

```
#include <iostream>
#include <boost/asio.hpp>
#include <boost/bind.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>
```

To implement a repeating timer using asio you need to change the timer's expiry time in your callback function, and to then start a new asynchronous wait. Obviously this means that the callback function will need to be able to access the timer object. To this end we add two new parameters to the `print` function:

- A pointer to a timer object.
- A counter so that we can stop the program when the timer fires for the sixth time.

```
void print(const boost::system::error_code& /*e*/,
    boost::asio::deadline_timer* t, int* count)
{
```

As mentioned above, this tutorial program uses a counter to stop running when the timer fires for the sixth time. However you will observe that there is no explicit call to ask the io_service to stop. Recall that in tutorial Timer.2 we learnt that the io_service::run() function completes when there is no more "work" to do. By not starting a new asynchronous wait on the timer when `count` reaches 5, the io_service will run out of work and stop running.

```
  if (*count < 5)
  {
    std::cout << *count << "\n";
    ++(*count);
```

Next we move the expiry time for the timer along by one second from the previous expiry time. By calculating the new expiry time relative to the old, we can ensure that the timer does not drift away from the whole-second mark due to any delays in processing the handler.

```
    t->expires_at(t->expires_at() + boost::posix_time::seconds(1));
```

Then we start a new asynchronous wait on the timer. As you can see, the boost::bind() function is used to associate the extra parameters with your callback handler. The deadline_timer::async_wait() function expects a handler function (or function object) with the signature `void(const boost::system::error_code&)`. Binding the additional parameters converts your `print` function into a function object that matches the signature correctly.

See the Boost.Bind documentation for more information on how to use boost::bind().

In this example, the boost::asio::placeholders::error argument to boost::bind() is a named placeholder for the error object passed to the handler. When initiating the asynchronous operation, and if using boost::bind(), you must specify only the arguments that match the handler's parameter list. In tutorial Timer.4 you will see that this placeholder may be elided if the parameter is not needed by the callback handler.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    t->async_wait(boost::bind(print,
          boost::asio::placeholders::error, t, count));
  }
}

int main()
{
  boost::asio::io_service io;
```

A new `count` variable is added so that we can stop the program when the timer fires for the sixth time.

```
  int count = 0;
  boost::asio::deadline_timer t(io, boost::posix_time::seconds(1));
```

As in Step 4, when making the call to deadline_timer::async_wait() from `main` we bind the additional parameters needed for the `print` function.

```
  t.async_wait(boost::bind(print,
        boost::asio::placeholders::error, &t, &count));

  io.run();
```

Finally, just to prove that the `count` variable was being used in the `print` handler function, we will print out its new value.

```
  std::cout << "Final count is " << count << "\n";

  return 0;
}
```

See the full source listing

Return to the tutorial index

Previous: Timer.2 - Using a timer asynchronously

Next: Timer.4 - Using a member function as a handler

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Source listing for Timer.3

```
//
// timer.cpp
// ~~~~~~~~~
//
// Copyright (c) 2003-2012 Christopher M. Kohlhoff (chris at kohlhoff dot com)
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//

#include <iostream>
#include <boost/asio.hpp>
#include <boost/bind.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>

void print(const boost::system::error_code& /*e*/,
    boost::asio::deadline_timer* t, int* count)
{
  if (*count < 5)
  {
    std::cout << *count << "\n";
    ++(*count);

    t->expires_at(t->expires_at() + boost::posix_time::seconds(1));
    t->async_wait(boost::bind(print,
          boost::asio::placeholders::error, t, count));
  }
}

int main()
{
  boost::asio::io_service io;

  int count = 0;
  boost::asio::deadline_timer t(io, boost::posix_time::seconds(1));
  t.async_wait(boost::bind(print,
        boost::asio::placeholders::error, &t, &count));

  io.run();

  std::cout << "Final count is " << count << "\n";

  return 0;
}
```

Return to Timer.3 - Binding arguments to a handler

Timer.4 - Using a member function as a handler

In this tutorial we will see how to use a class member function as a callback handler. The program should execute identically to the tutorial program from tutorial Timer.3.

```
#include <iostream>
#include <boost/asio.hpp>
#include <boost/bind.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Instead of defining a free function `print` as the callback handler, as we did in the earlier tutorial programs, we now define a class called `printer`.

```
class printer
{
public:
```

The constructor of this class will take a reference to the io_service object and use it when initialising the `timer_` member. The counter used to shut down the program is now also a member of the class.

```
  printer(boost::asio::io_service& io)
    : timer_(io, boost::posix_time::seconds(1)),
      count_(0)
  {
```

The boost::bind() function works just as well with class member functions as with free functions. Since all non-static class member functions have an implicit `this` parameter, we need to bind `this` to the function. As in tutorial Timer.3, boost::bind() converts our callback handler (now a member function) into a function object that can be invoked as though it has the signature `void(const boost::system::error_code&)`.

You will note that the boost::asio::placeholders::error placeholder is not specified here, as the `print` member function does not accept an error object as a parameter.

```
    timer_.async_wait(boost::bind(&printer::print, this));
  }
```

In the class destructor we will print out the final value of the counter.

```
  ~printer()
  {
    std::cout << "Final count is " << count_ << "\n";
  }
```

The `print` member function is very similar to the `print` function from tutorial Timer.3, except that it now operates on the class data members instead of having the timer and counter passed in as parameters.

```
  void print()
  {
    if (count_ < 5)
    {
      std::cout << count_ << "\n";
      ++count_;

      timer_.expires_at(timer_.expires_at() + boost::posix_time::seconds(1));
      timer_.async_wait(boost::bind(&printer::print, this));
    }
  }

private:
  boost::asio::deadline_timer timer_;
  int count_;
};
```

The `main` function is much simpler than before, as it now declares a local `printer` object before running the io_service as normal.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
int main()
{
  boost::asio::io_service io;
  printer p(io);
  io.run();

  return 0;
}
```

See the full source listing

Return to the tutorial index

Previous: Timer.3 - Binding arguments to a handler

Next: Timer.5 - Synchronising handlers in multithreaded programs

Source listing for Timer.4

```
//
// timer.cpp
// ~~~~~~~~~
//
// Copyright (c) 2003-2012 Christopher M. Kohlhoff (chris at kohlhoff dot com)
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//

#include <iostream>
#include <boost/asio.hpp>
#include <boost/bind.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>

class printer
{
public:
  printer(boost::asio::io_service& io)
    : timer_(io, boost::posix_time::seconds(1)),
      count_(0)
  {
    timer_.async_wait(boost::bind(&printer::print, this));
  }

  ~printer()
  {
    std::cout << "Final count is " << count_ << "\n";
  }

  void print()
  {
    if (count_ < 5)
    {
      std::cout << count_ << "\n";
      ++count_;

      timer_.expires_at(timer_.expires_at() + boost::posix_time::seconds(1));
      timer_.async_wait(boost::bind(&printer::print, this));
    }
  }
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
private:
  boost::asio::deadline_timer timer_;
  int count_;
};

int main()
{
  boost::asio::io_service io;
  printer p(io);
  io.run();

  return 0;
}
```

Return to Timer.4 - Using a member function as a handler

Timer.5 - Synchronising handlers in multithreaded programs

This tutorial demonstrates the use of the boost::asio::strand class to synchronise callback handlers in a multithreaded program.

The previous four tutorials avoided the issue of handler synchronisation by calling the io_service::run() function from one thread only. As you already know, the asio library provides a guarantee that callback handlers will only be called from threads that are currently calling io_service::run(). Consequently, calling io_service::run() from only one thread ensures that callback handlers cannot run concurrently.

The single threaded approach is usually the best place to start when developing applications using asio. The downside is the limitations it places on programs, particularly servers, including:

- Poor responsiveness when handlers can take a long time to complete.
- An inability to scale on multiprocessor systems.

If you find yourself running into these limitations, an alternative approach is to have a pool of threads calling io_service::run(). However, as this allows handlers to execute concurrently, we need a method of synchronisation when handlers might be accessing a shared, thread-unsafe resource.

```
#include <iostream>
#include <boost/asio.hpp>
#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>
```

We start by defining a class called `printer`, similar to the class in the previous tutorial. This class will extend the previous tutorial by running two timers in parallel.

```
class printer
{
public:
```

In addition to initialising a pair of boost::asio::deadline_timer members, the constructor initialises the `strand_` member, an object of type boost::asio::strand.

An boost::asio::strand guarantees that, for those handlers that are dispatched through it, an executing handler will be allowed to complete before the next one is started. This is guaranteed irrespective of the number of threads that are calling io_service::run(). Of course, the handlers may still execute concurrently with other handlers that were not dispatched through an boost::asio::strand, or were dispatched through a different boost::asio::strand object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  printer(boost::asio::io_service& io)
    : strand_(io),
      timer1_(io, boost::posix_time::seconds(1)),
      timer2_(io, boost::posix_time::seconds(1)),
      count_(0)
  {
```

When initiating the asynchronous operations, each callback handler is "wrapped" using the boost::asio::strand object. The strand::wrap() function returns a new handler that automatically dispatches its contained handler through the boost::asio::strand object. By wrapping the handlers using the same boost::asio::strand, we are ensuring that they cannot execute concurrently.

```
    timer1_.async_wait(strand_.wrap(boost::bind(&printer::print1, this)));
    timer2_.async_wait(strand_.wrap(boost::bind(&printer::print2, this)));
  }

  ~printer()
  {
    std::cout << "Final count is " << count_ << "\n";
  }
```

In a multithreaded program, the handlers for asynchronous operations should be synchronised if they access shared resources. In this tutorial, the shared resources used by the handlers (`print1` and `print2`) are `std::cout` and the `count_` data member.

```
  void print1()
  {
    if (count_ < 10)
    {
      std::cout << "Timer 1: " << count_ << "\n";
      ++count_;

      timer1_.expires_at(timer1_.expires_at() + boost::posix_time::seconds(1));
      timer1_.async_wait(strand_.wrap(boost::bind(&printer::print1, this)));
    }
  }

  void print2()
  {
    if (count_ < 10)
    {
      std::cout << "Timer 2: " << count_ << "\n";
      ++count_;

      timer2_.expires_at(timer2_.expires_at() + boost::posix_time::seconds(1));
      timer2_.async_wait(strand_.wrap(boost::bind(&printer::print2, this)));
    }
  }

private:
  boost::asio::strand strand_;
  boost::asio::deadline_timer timer1_;
  boost::asio::deadline_timer timer2_;
  int count_;
};
```

The `main` function now causes io_service::run() to be called from two threads: the main thread and one additional thread. This is accomplished using an boost::thread object.

Just as it would with a call from a single thread, concurrent calls to io_service::run() will continue to execute while there is "work" left to do. The background thread will not exit until all asynchronous operations have completed.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
int main()
{
  boost::asio::io_service io;
  printer p(io);
  boost::thread t(boost::bind(&boost::asio::io_service::run, &io));
  io.run();
  t.join();

  return 0;
}
```

See the full source listing

Return to the tutorial index

Previous: Timer.4 - Using a member function as a handler

Source listing for Timer.5

```
//
// timer.cpp
// ~~~~~~~~~
//
// Copyright (c) 2003-2012 Christopher M. Kohlhoff (chris at kohlhoff dot com)
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//

#include <iostream>
#include <boost/asio.hpp>
#include <boost/thread/thread.hpp>
#include <boost/bind.hpp>
#include <boost/date_time/posix_time/posix_time.hpp>

class printer
{
public:
  printer(boost::asio::io_service& io)
    : strand_(io),
      timer1_(io, boost::posix_time::seconds(1)),
      timer2_(io, boost::posix_time::seconds(1)),
      count_(0)
  {
    timer1_.async_wait(strand_.wrap(boost::bind(&printer::print1, this)));
    timer2_.async_wait(strand_.wrap(boost::bind(&printer::print2, this)));
  }

  ~printer()
  {
    std::cout << "Final count is " << count_ << "\n";
  }

  void print1()
  {
    if (count_ < 10)
    {
      std::cout << "Timer 1: " << count_ << "\n";
      ++count_;

      timer1_.expires_at(timer1_.expires_at() + boost::posix_time::seconds(1));
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
      timer1_.async_wait(strand_.wrap(boost::bind(&printer::print1, this)));
    }
  }

  void print2()
  {
    if (count_ < 10)
    {
      std::cout << "Timer 2: " << count_ << "\n";
      ++count_;

      timer2_.expires_at(timer2_.expires_at() + boost::posix_time::seconds(1));
      timer2_.async_wait(strand_.wrap(boost::bind(&printer::print2, this)));
    }
  }

private:
  boost::asio::strand strand_;
  boost::asio::deadline_timer timer1_;
  boost::asio::deadline_timer timer2_;
  int count_;
};

int main()
{
  boost::asio::io_service io;
  printer p(io);
  boost::thread t(boost::bind(&boost::asio::io_service::run, &io));
  io.run();
  t.join();

  return 0;
}
```

Return to Timer.5 - Synchronising handlers in multithreaded programs

Daytime.1 - A synchronous TCP daytime client

This tutorial program shows how to use asio to implement a client application with TCP.

We start by including the necessary header files.

```
#include <iostream>
#include <boost/array.hpp>
#include <boost/asio.hpp>
```

The purpose of this application is to access a daytime service, so we need the user to specify the server.

```
using boost::asio::ip::tcp;

int main(int argc, char* argv[])
{
  try
  {
    if (argc != 2)
    {
      std::cerr << "Usage: client <host>" << std::endl;
      return 1;
    }
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

All programs that use asio need to have at least one io_service object.

```
    boost::asio::io_service io_service;
```

We need to turn the server name that was specified as a parameter to the application, into a TCP endpoint. To do this we use an ip::tcp::resolver object.

```
    tcp::resolver resolver(io_service);
```

A resolver takes a query object and turns it into a list of endpoints. We construct a query using the name of the server, specified in `argv[1]`, and the name of the service, in this case `"daytime"`.

```
    tcp::resolver::query query(argv[1], "daytime");
```

The list of endpoints is returned using an iterator of type ip::tcp::resolver::iterator. (Note that a default constructed ip::tcp::resolver::iterator object can be used as an end iterator.)

```
    tcp::resolver::iterator endpoint_iterator = resolver.resolve(query);
```

Now we create and connect the socket. The list of endpoints obtained above may contain both IPv4 and IPv6 endpoints, so we need to try each of them until we find one that works. This keeps the client program independent of a specific IP version. The boost::asio::connect() function does this for us automatically.

```
    tcp::socket socket(io_service);
    boost::asio::connect(socket, endpoint_iterator);
```

The connection is open. All we need to do now is read the response from the daytime service.

We use a `boost::array` to hold the received data. The boost::asio::buffer() function automatically determines the size of the array to help prevent buffer overruns. Instead of a `boost::array`, we could have used a `char []` or `std::vector`.

```
    for (;;)
    {
      boost::array<char, 128> buf;
      boost::system::error_code error;

      size_t len = socket.read_some(boost::asio::buffer(buf), error);
```

When the server closes the connection, the ip::tcp::socket::read_some() function will exit with the boost::asio::error::eof error, which is how we know to exit the loop.

```
      if (error == boost::asio::error::eof)
        break; // Connection closed cleanly by peer.
      else if (error)
        throw boost::system::system_error(error); // Some other error.

      std::cout.write(buf.data(), len);
    }
```

Finally, handle any exceptions that may have been thrown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  }
  catch (std::exception& e)
  {
    std::cerr << e.what() << std::endl;
  }
```

See the full source listing

Return to the tutorial index

Next: Daytime.2 - A synchronous TCP daytime server

Source listing for Daytime.1

```
//
// client.cpp
// ~~~~~~~~~~
//
// Copyright (c) 2003-2012 Christopher M. Kohlhoff (chris at kohlhoff dot com)
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//

#include <iostream>
#include <boost/array.hpp>
#include <boost/asio.hpp>

using boost::asio::ip::tcp;

int main(int argc, char* argv[])
{
  try
  {
    if (argc != 2)
    {
      std::cerr << "Usage: client <host>" << std::endl;
      return 1;
    }

    boost::asio::io_service io_service;

    tcp::resolver resolver(io_service);
    tcp::resolver::query query(argv[1], "daytime");
    tcp::resolver::iterator endpoint_iterator = resolver.resolve(query);

    tcp::socket socket(io_service);
    boost::asio::connect(socket, endpoint_iterator);

    for (;;)
    {
      boost::array<char, 128> buf;
      boost::system::error_code error;

      size_t len = socket.read_some(boost::asio::buffer(buf), error);

      if (error == boost::asio::error::eof)
        break; // Connection closed cleanly by peer.
      else if (error)
        throw boost::system::system_error(error); // Some other error.
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
      std::cout.write(buf.data(), len);
    }
  }
  catch (std::exception& e)
  {
    std::cerr << e.what() << std::endl;
  }

  return 0;
}
```

Return to Daytime.1 - A synchronous TCP daytime client

Daytime.2 - A synchronous TCP daytime server

This tutorial program shows how to use asio to implement a server application with TCP.

```
#include <ctime>
#include <iostream>
#include <string>
#include <boost/asio.hpp>

using boost::asio::ip::tcp;
```

We define the function `make_daytime_string()` to create the string to be sent back to the client. This function will be reused in all of our daytime server applications.

```
std::string make_daytime_string()
{
  using namespace std; // For time_t, time and ctime;
  time_t now = time(0);
  return ctime(&now);
}

int main()
{
  try
  {
    boost::asio::io_service io_service;
```

A ip::tcp::acceptor object needs to be created to listen for new connections. It is initialised to listen on TCP port 13, for IP version 4.

```
    tcp::acceptor acceptor(io_service, tcp::endpoint(tcp::v4(), 13));
```

This is an iterative server, which means that it will handle one connection at a time. Create a socket that will represent the connection to the client, and then wait for a connection.

```
    for (;;)
    {
      tcp::socket socket(io_service);
      acceptor.accept(socket);
```

A client is accessing our service. Determine the current time and transfer this information to the client.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
      std::string message = make_daytime_string();

      boost::system::error_code ignored_error;
      boost::asio::write(socket, boost::asio::buffer(message), ignored_error);
    }
  }
```

Finally, handle any exceptions.

```
  catch (std::exception& e)
  {
    std::cerr << e.what() << std::endl;
  }

  return 0;
}
```

See the full source listing

Return to the tutorial index

Previous: Daytime.1 - A synchronous TCP daytime client

Next: Daytime.3 - An asynchronous TCP daytime server

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Source listing for Daytime.2

```
//
// server.cpp
// ~~~~~~~~~~
//
// Copyright (c) 2003-2012 Christopher M. Kohlhoff (chris at kohlhoff dot com)
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//

#include <ctime>
#include <iostream>
#include <string>
#include <boost/asio.hpp>

using boost::asio::ip::tcp;

std::string make_daytime_string()
{
  using namespace std; // For time_t, time and ctime;
  time_t now = time(0);
  return ctime(&now);
}

int main()
{
  try
  {
    boost::asio::io_service io_service;

    tcp::acceptor acceptor(io_service, tcp::endpoint(tcp::v4(), 13));

    for (;;)
    {
      tcp::socket socket(io_service);
      acceptor.accept(socket);

      std::string message = make_daytime_string();

      boost::system::error_code ignored_error;
      boost::asio::write(socket, boost::asio::buffer(message), ignored_error);
    }
  }
  catch (std::exception& e)
  {
    std::cerr << e.what() << std::endl;
  }

  return 0;
}
```

Return to Daytime.2 - A synchronous TCP daytime server

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Daytime.3 - An asynchronous TCP daytime server

The main() function

```
int main()
{
  try
  {
```

We need to create a server object to accept incoming client connections. The io_service object provides I/O services, such as sockets, that the server object will use.

```
    boost::asio::io_service io_service;
    tcp_server server(io_service);
```

Run the io_service object so that it will perform asynchronous operations on your behalf.

```
    io_service.run();
  }
  catch (std::exception& e)
  {
    std::cerr << e.what() << std::endl;
  }

  return 0;
}
```

The tcp_server class

```
class tcp_server
{
public:
```

The constructor initialises an acceptor to listen on TCP port 13.

```
  tcp_server(boost::asio::io_service& io_service)
    : acceptor_(io_service, tcp::endpoint(tcp::v4(), 13))
  {
    start_accept();
  }

private:
```

The function `start_accept()` creates a socket and initiates an asynchronous accept operation to wait for a new connection.

```
  void start_accept()
  {
    tcp_connection::pointer new_connection =
      tcp_connection::create(acceptor_.get_io_service());

    acceptor_.async_accept(new_connection->socket(),
        boost::bind(&tcp_server::handle_accept, this, new_connection,
          boost::asio::placeholders::error));
  }
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The function `handle_accept()` is called when the asynchronous accept operation initiated by `start_accept()` finishes. It services the client request, and then calls `start_accept()` to initiate the next accept operation.

```
  void handle_accept(tcp_connection::pointer new_connection,
      const boost::system::error_code& error)
  {
    if (!error)
    {
      new_connection->start();
    }

    start_accept();
  }
```

The tcp_connection class

We will use `shared_ptr` and `enable_shared_from_this` because we want to keep the `tcp_connection` object alive as long as there is an operation that refers to it.

```
class tcp_connection
  : public boost::enable_shared_from_this<tcp_connection>
{
public:
  typedef boost::shared_ptr<tcp_connection> pointer;

  static pointer create(boost::asio::io_service& io_service)
  {
    return pointer(new tcp_connection(io_service));
  }

  tcp::socket& socket()
  {
    return socket_;
  }
```

In the function `start()`, we call boost::asio::async_write() to serve the data to the client. Note that we are using boost::asio::async_write(), rather than ip::tcp::socket::async_write_some(), to ensure that the entire block of data is sent.

```
  void start()
  {
```

The data to be sent is stored in the class member `message_` as we need to keep the data valid until the asynchronous operation is complete.

```
    message_ = make_daytime_string();
```

When initiating the asynchronous operation, and if using boost::bind(), you must specify only the arguments that match the handler's parameter list. In this program, both of the argument placeholders (boost::asio::placeholders::error and boost::asio::placeholders::bytes_transferred) could potentially have been removed, since they are not being used in `handle_write()`.

```
    boost::asio::async_write(socket_, boost::asio::buffer(message_),
        boost::bind(&tcp_connection::handle_write, shared_from_this(),
          boost::asio::placeholders::error,
          boost::asio::placeholders::bytes_transferred));
```

Any further actions for this client connection are now the responsibility of `handle_write()`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  }

private:
  tcp_connection(boost::asio::io_service& io_service)
    : socket_(io_service)
  {
  }

  void handle_write(const boost::system::error_code& /*error*/,
      size_t /*bytes_transferred*/)
  {
  }

  tcp::socket socket_;
  std::string message_;
};
```

Removing unused handler parameters

You may have noticed that the `error`, and `bytes_transferred` parameters are not used in the body of the `handle_write()` function. If parameters are not needed, it is possible to remove them from the function so that it looks like:

```
  void handle_write()
  {
  }
```

The boost::asio::async_write() call used to initiate the call can then be changed to just:

```
  boost::asio::async_write(socket_, boost::asio::buffer(message_),
      boost::bind(&tcp_connection::handle_write, shared_from_this()));
```

See the full source listing

Return to the tutorial index

Previous: Daytime.2 - A synchronous TCP daytime server

Next: Daytime.4 - A synchronous UDP daytime client

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Source listing for Daytime.3

```
//
// server.cpp
// ~~~~~~~~~~
//
// Copyright (c) 2003-2012 Christopher M. Kohlhoff (chris at kohlhoff dot com)
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//

#include <ctime>
#include <iostream>
#include <string>
#include <boost/bind.hpp>
#include <boost/shared_ptr.hpp>
#include <boost/enable_shared_from_this.hpp>
#include <boost/asio.hpp>

using boost::asio::ip::tcp;

std::string make_daytime_string()
{
  using namespace std; // For time_t, time and ctime;
  time_t now = time(0);
  return ctime(&now);
}

class tcp_connection
  : public boost::enable_shared_from_this<tcp_connection>
{
public:
  typedef boost::shared_ptr<tcp_connection> pointer;

  static pointer create(boost::asio::io_service& io_service)
  {
    return pointer(new tcp_connection(io_service));
  }

  tcp::socket& socket()
  {
    return socket_;
  }

  void start()
  {
    message_ = make_daytime_string();

    boost::asio::async_write(socket_, boost::asio::buffer(message_),
        boost::bind(&tcp_connection::handle_write, shared_from_this(),
          boost::asio::placeholders::error,
          boost::asio::placeholders::bytes_transferred));
  }

private:
  tcp_connection(boost::asio::io_service& io_service)
    : socket_(io_service)
  {
  }

  void handle_write(const boost::system::error_code& /*error*/,
      size_t /*bytes_transferred*/)
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  {
  }

  tcp::socket socket_;
  std::string message_;
};

class tcp_server
{
public:
  tcp_server(boost::asio::io_service& io_service)
    : acceptor_(io_service, tcp::endpoint(tcp::v4(), 13))
  {
    start_accept();
  }

private:
  void start_accept()
  {
    tcp_connection::pointer new_connection =
      tcp_connection::create(acceptor_.get_io_service());

    acceptor_.async_accept(new_connection->socket(),
        boost::bind(&tcp_server::handle_accept, this, new_connection,
          boost::asio::placeholders::error));
  }

  void handle_accept(tcp_connection::pointer new_connection,
      const boost::system::error_code& error)
  {
    if (!error)
    {
      new_connection->start();
    }

    start_accept();
  }

  tcp::acceptor acceptor_;
};

int main()
{
  try
  {
    boost::asio::io_service io_service;
    tcp_server server(io_service);
    io_service.run();
  }
  catch (std::exception& e)
  {
    std::cerr << e.what() << std::endl;
  }

  return 0;
}
```

Return to Daytime.3 - An asynchronous TCP daytime server

Daytime.4 - A synchronous UDP daytime client

This tutorial program shows how to use asio to implement a client application with UDP.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
#include <iostream>
#include <boost/array.hpp>
#include <boost/asio.hpp>

using boost::asio::ip::udp;
```

The start of the application is essentially the same as for the TCP daytime client.

```
int main(int argc, char* argv[])
{
  try
  {
    if (argc != 2)
    {
      std::cerr << "Usage: client <host>" << std::endl;
      return 1;
    }

    boost::asio::io_service io_service;
```

We use an ip::udp::resolver object to find the correct remote endpoint to use based on the host and service names. The query is restricted to return only IPv4 endpoints by the ip::udp::v4() argument.

```
    udp::resolver resolver(io_service);
    udp::resolver::query query(udp::v4(), argv[1], "daytime");
```

The ip::udp::resolver::resolve() function is guaranteed to return at least one endpoint in the list if it does not fail. This means it is safe to dereference the return value directly.

```
    udp::endpoint receiver_endpoint = *resolver.resolve(query);
```

Since UDP is datagram-oriented, we will not be using a stream socket. Create an ip::udp::socket and initiate contact with the remote endpoint.

```
    udp::socket socket(io_service);
    socket.open(udp::v4());

    boost::array<char, 1> send_buf  = {{ 0 }};
    socket.send_to(boost::asio::buffer(send_buf), receiver_endpoint);
```

Now we need to be ready to accept whatever the server sends back to us. The endpoint on our side that receives the server's response will be initialised by ip::udp::socket::receive_from().

```
    boost::array<char, 128> recv_buf;
    udp::endpoint sender_endpoint;
    size_t len = socket.receive_from(
        boost::asio::buffer(recv_buf), sender_endpoint);

    std::cout.write(recv_buf.data(), len);
  }
```

Finally, handle any exceptions that may have been thrown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  catch (std::exception& e)
  {
    std::cerr << e.what() << std::endl;
  }

  return 0;
}
```

See the full source listing

Return to the tutorial index

Previous: Daytime.3 - An asynchronous TCP daytime server

Next: Daytime.5 - A synchronous UDP daytime server

Source listing for Daytime.4

```
//
// client.cpp
// ~~~~~~~~~~
//
// Copyright (c) 2003-2012 Christopher M. Kohlhoff (chris at kohlhoff dot com)
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//

#include <iostream>
#include <boost/array.hpp>
#include <boost/asio.hpp>

using boost::asio::ip::udp;

int main(int argc, char* argv[])
{
  try
  {
    if (argc != 2)
    {
      std::cerr << "Usage: client <host>" << std::endl;
      return 1;
    }

    boost::asio::io_service io_service;

    udp::resolver resolver(io_service);
    udp::resolver::query query(udp::v4(), argv[1], "daytime");
    udp::endpoint receiver_endpoint = *resolver.resolve(query);

    udp::socket socket(io_service);
    socket.open(udp::v4());

    boost::array<char, 1> send_buf  = {{ 0 }};
    socket.send_to(boost::asio::buffer(send_buf), receiver_endpoint);

    boost::array<char, 128> recv_buf;
    udp::endpoint sender_endpoint;
    size_t len = socket.receive_from(
        boost::asio::buffer(recv_buf), sender_endpoint);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    std::cout.write(recv_buf.data(), len);
  }
  catch (std::exception& e)
  {
    std::cerr << e.what() << std::endl;
  }

  return 0;
}
```

Return to Daytime.4 - A synchronous UDP daytime client

Daytime.5 - A synchronous UDP daytime server

This tutorial program shows how to use asio to implement a server application with UDP.

```
int main()
{
  try
  {
    boost::asio::io_service io_service;
```

Create an ip::udp::socket object to receive requests on UDP port 13.

```
    udp::socket socket(io_service, udp::endpoint(udp::v4(), 13));
```

Wait for a client to initiate contact with us. The remote_endpoint object will be populated by ip::udp::socket::receive_from().

```
    for (;;)
    {
      boost::array<char, 1> recv_buf;
      udp::endpoint remote_endpoint;
      boost::system::error_code error;
      socket.receive_from(boost::asio::buffer(recv_buf),
          remote_endpoint, 0, error);

      if (error && error != boost::asio::error::message_size)
        throw boost::system::system_error(error);
```

Determine what we are going to send back to the client.

```
      std::string message = make_daytime_string();
```

Send the response to the remote_endpoint.

```
      boost::system::error_code ignored_error;
      socket.send_to(boost::asio::buffer(message),
          remote_endpoint, 0, ignored_error);
    }
  }
```

Finally, handle any exceptions.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  catch (std::exception& e)
  {
    std::cerr << e.what() << std::endl;
  }

  return 0;
}
```

See the full source listing

Return to the tutorial index

Previous: Daytime.4 - A synchronous UDP daytime client

Next: Daytime.6 - An asynchronous UDP daytime server

Source listing for Daytime.5

```
//
// server.cpp
// ~~~~~~~~~~
//
// Copyright (c) 2003-2012 Christopher M. Kohlhoff (chris at kohlhoff dot com)
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//

#include <ctime>
#include <iostream>
#include <string>
#include <boost/array.hpp>
#include <boost/asio.hpp>

using boost::asio::ip::udp;

std::string make_daytime_string()
{
  using namespace std; // For time_t, time and ctime;
  time_t now = time(0);
  return ctime(&now);
}

int main()
{
  try
  {
    boost::asio::io_service io_service;

    udp::socket socket(io_service, udp::endpoint(udp::v4(), 13));

    for (;;)
    {
      boost::array<char, 1> recv_buf;
      udp::endpoint remote_endpoint;
      boost::system::error_code error;
      socket.receive_from(boost::asio::buffer(recv_buf),
          remote_endpoint, 0, error);

      if (error && error != boost::asio::error::message_size)
        throw boost::system::system_error(error);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
      std::string message = make_daytime_string();

      boost::system::error_code ignored_error;
      socket.send_to(boost::asio::buffer(message),
          remote_endpoint, 0, ignored_error);
    }
  }
  catch (std::exception& e)
  {
    std::cerr << e.what() << std::endl;
  }

  return 0;
}
```

Return to Daytime.5 - A synchronous UDP daytime server

Daytime.6 - An asynchronous UDP daytime server

The main() function

```
int main()
{
  try
  {
```

Create a server object to accept incoming client requests, and run the io_service object.

```
    boost::asio::io_service io_service;
    udp_server server(io_service);
    io_service.run();
  }
  catch (std::exception& e)
  {
    std::cerr << e.what() << std::endl;
  }

  return 0;
}
```

The udp_server class

```
class udp_server
{
public:
```

The constructor initialises a socket to listen on UDP port 13.

```
  udp_server(boost::asio::io_service& io_service)
    : socket_(io_service, udp::endpoint(udp::v4(), 13))
  {
    start_receive();
  }

private:
  void start_receive()
  {
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The function ip::udp::socket::async_receive_from() will cause the application to listen in the background for a new request. When such a request is received, the io_service object will invoke the `handle_receive()` function with two arguments: a value of type boost::system::error_code indicating whether the operation succeeded or failed, and a `size_t` value `bytes_transferred` specifying the number of bytes received.

```
    socket_.async_receive_from(
        boost::asio::buffer(recv_buffer_), remote_endpoint_,
        boost::bind(&udp_server::handle_receive, this,
          boost::asio::placeholders::error,
          boost::asio::placeholders::bytes_transferred));
  }
```

The function `handle_receive()` will service the client request.

```
  void handle_receive(const boost::system::error_code& error,
      std::size_t /*bytes_transferred*/)
  {
```

The `error` parameter contains the result of the asynchronous operation. Since we only provide the 1-byte `recv_buffer_` to contain the client's request, the io_service object would return an error if the client sent anything larger. We can ignore such an error if it comes up.

```
    if (!error || error == boost::asio::error::message_size)
    {
```

Determine what we are going to send.

```
      boost::shared_ptr<std::string> message(
          new std::string(make_daytime_string()));
```

We now call ip::udp::socket::async_send_to() to serve the data to the client.

```
      socket_.async_send_to(boost::asio::buffer(*message), remote_endpoint_,
          boost::bind(&udp_server::handle_send, this, message,
            boost::asio::placeholders::error,
            boost::asio::placeholders::bytes_transferred));
```

When initiating the asynchronous operation, and if using boost::bind(), you must specify only the arguments that match the handler's parameter list. In this program, both of the argument placeholders (boost::asio::placeholders::error and boost::asio::placeholders::bytes_transferred) could potentially have been removed.

Start listening for the next client request.

```
      start_receive();
```

Any further actions for this client request are now the responsibility of `handle_send()`.

```
    }
  }
```

The function `handle_send()` is invoked after the service request has been completed.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  void handle_send(boost::shared_ptr<std::string> /*message*/,
      const boost::system::error_code& /*error*/,
      std::size_t /*bytes_transferred*/)
  {
  }

  udp::socket socket_;
  udp::endpoint remote_endpoint_;
  boost::array<char, 1> recv_buffer_;
};
```

See the full source listing

Return to the tutorial index

Previous: Daytime.5 - A synchronous UDP daytime server

Next: Daytime.7 - A combined TCP/UDP asynchronous server

Source listing for Daytime.6

```
//
// server.cpp
// ~~~~~~~~~~
//
// Copyright (c) 2003-2012 Christopher M. Kohlhoff (chris at kohlhoff dot com)
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//

#include <ctime>
#include <iostream>
#include <string>
#include <boost/array.hpp>
#include <boost/bind.hpp>
#include <boost/shared_ptr.hpp>
#include <boost/asio.hpp>

using boost::asio::ip::udp;

std::string make_daytime_string()
{
  using namespace std; // For time_t, time and ctime;
  time_t now = time(0);
  return ctime(&now);
}

class udp_server
{
public:
  udp_server(boost::asio::io_service& io_service)
    : socket_(io_service, udp::endpoint(udp::v4(), 13))
  {
    start_receive();
  }

private:
  void start_receive()
  {
    socket_.async_receive_from(
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
        boost::asio::buffer(recv_buffer_), remote_endpoint_,
        boost::bind(&udp_server::handle_receive, this,
          boost::asio::placeholders::error,
          boost::asio::placeholders::bytes_transferred));
  }

  void handle_receive(const boost::system::error_code& error,
      std::size_t /*bytes_transferred*/)
  {
    if (!error || error == boost::asio::error::message_size)
    {
      boost::shared_ptr<std::string> message(
          new std::string(make_daytime_string()));

      socket_.async_send_to(boost::asio::buffer(*message), remote_endpoint_,
          boost::bind(&udp_server::handle_send, this, message,
            boost::asio::placeholders::error,
            boost::asio::placeholders::bytes_transferred));

      start_receive();
    }
  }

  void handle_send(boost::shared_ptr<std::string> /*message*/,
      const boost::system::error_code& /*error*/,
      std::size_t /*bytes_transferred*/)
  {
  }

  udp::socket socket_;
  udp::endpoint remote_endpoint_;
  boost::array<char, 1> recv_buffer_;
};

int main()
{
  try
  {
    boost::asio::io_service io_service;
    udp_server server(io_service);
    io_service.run();
  }
  catch (std::exception& e)
  {
    std::cerr << e.what() << std::endl;
  }

  return 0;
}
```

Return to Daytime.6 - An asynchronous UDP daytime server

Daytime.7 - A combined TCP/UDP asynchronous server

This tutorial program shows how to combine the two asynchronous servers that we have just written, into a single server application.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The main() function

```
int main()
{
  try
  {
    boost::asio::io_service io_service;
```

We will begin by creating a server object to accept a TCP client connection.

```
    tcp_server server1(io_service);
```

We also need a server object to accept a UDP client request.

```
    udp_server server2(io_service);
```

We have created two lots of work for the io_service object to do.

```
    io_service.run();
  }
  catch (std::exception& e)
  {
    std::cerr << e.what() << std::endl;
  }

  return 0;
}
```

The tcp_connection and tcp_server classes

The following two classes are taken from Daytime.3 .

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
class tcp_connection
  : public boost::enable_shared_from_this<tcp_connection>
{
public:
  typedef boost::shared_ptr<tcp_connection> pointer;

  static pointer create(boost::asio::io_service& io_service)
  {
    return pointer(new tcp_connection(io_service));
  }

  tcp::socket& socket()
  {
    return socket_;
  }

  void start()
  {
    message_ = make_daytime_string();

    boost::asio::async_write(socket_, boost::asio::buffer(message_),
        boost::bind(&tcp_connection::handle_write, shared_from_this()));
  }

private:
  tcp_connection(boost::asio::io_service& io_service)
    : socket_(io_service)
  {
  }

  void handle_write()
  {
  }

  tcp::socket socket_;
  std::string message_;
};

class tcp_server
{
public:
  tcp_server(boost::asio::io_service& io_service)
    : acceptor_(io_service, tcp::endpoint(tcp::v4(), 13))
  {
    start_accept();
  }

private:
  void start_accept()
  {
    tcp_connection::pointer new_connection =
      tcp_connection::create(acceptor_.get_io_service());

    acceptor_.async_accept(new_connection->socket(),
        boost::bind(&tcp_server::handle_accept, this, new_connection,
          boost::asio::placeholders::error));
  }

  void handle_accept(tcp_connection::pointer new_connection,
      const boost::system::error_code& error)
  {
    if (!error)
    {
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
      new_connection->start();
    }

    start_accept();
  }

  tcp::acceptor acceptor_;
};
```

The udp_server class

Similarly, this next class is taken from the previous tutorial step .

```
class udp_server
{
public:
  udp_server(boost::asio::io_service& io_service)
    : socket_(io_service, udp::endpoint(udp::v4(), 13))
  {
    start_receive();
  }

private:
  void start_receive()
  {
    socket_.async_receive_from(
        boost::asio::buffer(recv_buffer_), remote_endpoint_,
        boost::bind(&udp_server::handle_receive, this,
          boost::asio::placeholders::error));
  }

  void handle_receive(const boost::system::error_code& error)
  {
    if (!error || error == boost::asio::error::message_size)
    {
      boost::shared_ptr<std::string> message(
          new std::string(make_daytime_string()));

      socket_.async_send_to(boost::asio::buffer(*message), remote_endpoint_,
          boost::bind(&udp_server::handle_send, this, message));

      start_receive();
    }
  }

  void handle_send(boost::shared_ptr<std::string> /*message*/)
  {
  }

  udp::socket socket_;
  udp::endpoint remote_endpoint_;
  boost::array<char, 1> recv_buffer_;
};
```

See the full source listing

Return to the tutorial index

Previous: Daytime.6 - An asynchronous UDP daytime server

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Source listing for Daytime.7

```
//
// server.cpp
// ~~~~~~~~~~
//
// Copyright (c) 2003-2012 Christopher M. Kohlhoff (chris at kohlhoff dot com)
//
// Distributed under the Boost Software License, Version 1.0. (See accompanying
// file LICENSE_1_0.txt or copy at http://www.boost.org/LICENSE_1_0.txt)
//

#include <ctime>
#include <iostream>
#include <string>
#include <boost/array.hpp>
#include <boost/bind.hpp>
#include <boost/shared_ptr.hpp>
#include <boost/enable_shared_from_this.hpp>
#include <boost/asio.hpp>

using boost::asio::ip::tcp;
using boost::asio::ip::udp;

std::string make_daytime_string()
{
  using namespace std; // For time_t, time and ctime;
  time_t now = time(0);
  return ctime(&now);
}

class tcp_connection
  : public boost::enable_shared_from_this<tcp_connection>
{
public:
  typedef boost::shared_ptr<tcp_connection> pointer;

  static pointer create(boost::asio::io_service& io_service)
  {
    return pointer(new tcp_connection(io_service));
  }

  tcp::socket& socket()
  {
    return socket_;
  }

  void start()
  {
    message_ = make_daytime_string();

    boost::asio::async_write(socket_, boost::asio::buffer(message_),
        boost::bind(&tcp_connection::handle_write, shared_from_this()));
  }

private:
  tcp_connection(boost::asio::io_service& io_service)
    : socket_(io_service)
  {
  }

  void handle_write()
  {
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  }

  tcp::socket socket_;
  std::string message_;
};

class tcp_server
{
public:
  tcp_server(boost::asio::io_service& io_service)
    : acceptor_(io_service, tcp::endpoint(tcp::v4(), 13))
  {
    start_accept();
  }

private:
  void start_accept()
  {
    tcp_connection::pointer new_connection =
      tcp_connection::create(acceptor_.get_io_service());

    acceptor_.async_accept(new_connection->socket(),
        boost::bind(&tcp_server::handle_accept, this, new_connection,
          boost::asio::placeholders::error));
  }

  void handle_accept(tcp_connection::pointer new_connection,
      const boost::system::error_code& error)
  {
    if (!error)
    {
      new_connection->start();
    }

    start_accept();
  }

  tcp::acceptor acceptor_;
};

class udp_server
{
public:
  udp_server(boost::asio::io_service& io_service)
    : socket_(io_service, udp::endpoint(udp::v4(), 13))
  {
    start_receive();
  }

private:
  void start_receive()
  {
    socket_.async_receive_from(
        boost::asio::buffer(recv_buffer_), remote_endpoint_,
        boost::bind(&udp_server::handle_receive, this,
          boost::asio::placeholders::error));
  }

  void handle_receive(const boost::system::error_code& error)
  {
    if (!error || error == boost::asio::error::message_size)
    {
      boost::shared_ptr<std::string> message(
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
          new std::string(make_daytime_string()));

      socket_.async_send_to(boost::asio::buffer(*message), remote_endpoint_,
          boost::bind(&udp_server::handle_send, this, message));

      start_receive();
    }
  }

  void handle_send(boost::shared_ptr<std::string> /*message*/)
  {
  }

  udp::socket socket_;
  udp::endpoint remote_endpoint_;
  boost::array<char, 1> recv_buffer_;
};

int main()
{
  try
  {
    boost::asio::io_service io_service;
    tcp_server server1(io_service);
    udp_server server2(io_service);
    io_service.run();
  }
  catch (std::exception& e)
  {
    std::cerr << e.what() << std::endl;
  }

  return 0;
}
```

Return to Daytime.7 - A combined TCP/UDP asynchronous server

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Examples

Allocation

This example shows how to customise the allocation of memory associated with asynchronous operations.

- boost_asio/example/allocation/server.cpp

Buffers

This example demonstrates how to create reference counted buffers that can be used with socket read and write operations.

- boost_asio/example/buffers/reference_counted.cpp

Chat

This example implements a chat server and client. The programs use a custom protocol with a fixed length message header and variable length message body.

- boost_asio/example/chat/chat_message.hpp
- boost_asio/example/chat/chat_client.cpp
- boost_asio/example/chat/chat_server.cpp

The following POSIX-specific chat client demonstrates how to use the posix::stream_descriptor class to perform console input and output.

- boost_asio/example/chat/posix_chat_client.cpp

Echo

A collection of simple clients and servers, showing the use of both synchronous and asynchronous operations.

- boost_asio/example/echo/async_tcp_echo_server.cpp
- boost_asio/example/echo/async_udp_echo_server.cpp
- boost_asio/example/echo/blocking_tcp_echo_client.cpp
- boost_asio/example/echo/blocking_tcp_echo_server.cpp
- boost_asio/example/echo/blocking_udp_echo_client.cpp
- boost_asio/example/echo/blocking_udp_echo_server.cpp

Fork

These POSIX-specific examples show how to use Boost.Asio in conjunction with the `fork()` system call. The first example illustrates the steps required to start a daemon process:

- boost_asio/example/fork/daemon.cpp

The second example demonstrates how it is possible to fork a process from within a completion handler.

- boost_asio/example/fork/process_per_connection.cpp

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

HTTP Client

Example programs implementing simple HTTP 1.0 clients. These examples show how to use the read_until and async_read_until functions.

- boost_asio/example/http/client/sync_client.cpp
- boost_asio/example/http/client/async_client.cpp

HTTP Server

This example illustrates the use of asio in a simple single-threaded server implementation of HTTP 1.0. It demonstrates how to perform a clean shutdown by cancelling all outstanding asynchronous operations.

- boost_asio/example/http/server/connection.cpp
- boost_asio/example/http/server/connection.hpp
- boost_asio/example/http/server/connection_manager.cpp
- boost_asio/example/http/server/connection_manager.hpp
- boost_asio/example/http/server/header.hpp
- boost_asio/example/http/server/main.cpp
- boost_asio/example/http/server/mime_types.cpp
- boost_asio/example/http/server/mime_types.hpp
- boost_asio/example/http/server/reply.cpp
- boost_asio/example/http/server/reply.hpp
- boost_asio/example/http/server/request.hpp
- boost_asio/example/http/server/request_handler.cpp
- boost_asio/example/http/server/request_handler.hpp
- boost_asio/example/http/server/request_parser.cpp
- boost_asio/example/http/server/request_parser.hpp
- boost_asio/example/http/server/server.cpp
- boost_asio/example/http/server/server.hpp

HTTP Server 2

An HTTP server using an io_service-per-CPU design.

- boost_asio/example/http/server2/connection.cpp
- boost_asio/example/http/server2/connection.hpp
- boost_asio/example/http/server2/header.hpp
- boost_asio/example/http/server2/io_service_pool.cpp
- boost_asio/example/http/server2/io_service_pool.hpp

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- boost_asio/example/http/server2/main.cpp
- boost_asio/example/http/server2/mime_types.cpp
- boost_asio/example/http/server2/mime_types.hpp
- boost_asio/example/http/server2/reply.cpp
- boost_asio/example/http/server2/reply.hpp
- boost_asio/example/http/server2/request.hpp
- boost_asio/example/http/server2/request_handler.cpp
- boost_asio/example/http/server2/request_handler.hpp
- boost_asio/example/http/server2/request_parser.cpp
- boost_asio/example/http/server2/request_parser.hpp
- boost_asio/example/http/server2/server.cpp
- boost_asio/example/http/server2/server.hpp

HTTP Server 3

An HTTP server using a single io_service and a thread pool calling `io_service::run()`.

- boost_asio/example/http/server3/connection.cpp
- boost_asio/example/http/server3/connection.hpp
- boost_asio/example/http/server3/header.hpp
- boost_asio/example/http/server3/main.cpp
- boost_asio/example/http/server3/mime_types.cpp
- boost_asio/example/http/server3/mime_types.hpp
- boost_asio/example/http/server3/reply.cpp
- boost_asio/example/http/server3/reply.hpp
- boost_asio/example/http/server3/request.hpp
- boost_asio/example/http/server3/request_handler.cpp
- boost_asio/example/http/server3/request_handler.hpp
- boost_asio/example/http/server3/request_parser.cpp
- boost_asio/example/http/server3/request_parser.hpp
- boost_asio/example/http/server3/server.cpp
- boost_asio/example/http/server3/server.hpp

HTTP Server 4

A single-threaded HTTP server implemented using stackless coroutines.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- boost_asio/example/http/server4/coroutine.hpp
- boost_asio/example/http/server4/file_handler.cpp
- boost_asio/example/http/server4/file_handler.hpp
- boost_asio/example/http/server4/header.hpp
- boost_asio/example/http/server4/main.cpp
- boost_asio/example/http/server4/mime_types.cpp
- boost_asio/example/http/server4/mime_types.hpp
- boost_asio/example/http/server4/reply.cpp
- boost_asio/example/http/server4/reply.hpp
- boost_asio/example/http/server4/request.hpp
- boost_asio/example/http/server4/request_parser.cpp
- boost_asio/example/http/server4/request_parser.hpp
- boost_asio/example/http/server4/server.cpp
- boost_asio/example/http/server4/server.hpp
- boost_asio/example/http/server4/unyield.hpp
- boost_asio/example/http/server4/yield.hpp

ICMP

This example shows how to use raw sockets with ICMP to ping a remote host.

- boost_asio/example/icmp/ping.cpp
- boost_asio/example/icmp/ipv4_header.hpp
- boost_asio/example/icmp/icmp_header.hpp

Invocation

This example shows how to customise handler invocation. Completion handlers are added to a priority queue rather than executed immediately.

- boost_asio/example/invocation/prioritised_handlers.cpp

Iostreams

Two examples showing how to use ip::tcp::iostream.

- boost_asio/example/iostreams/daytime_client.cpp
- boost_asio/example/iostreams/daytime_server.cpp
- boost_asio/example/iostreams/http_client.cpp

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Multicast

An example showing the use of multicast to transmit packets to a group of subscribers.

- boost_asio/example/multicast/receiver.cpp
- boost_asio/example/multicast/sender.cpp

Serialization

This example shows how Boost.Serialization can be used with asio to encode and decode structures for transmission over a socket.

- boost_asio/example/serialization/client.cpp
- boost_asio/example/serialization/connection.hpp
- boost_asio/example/serialization/server.cpp
- boost_asio/example/serialization/stock.hpp

Services

This example demonstrates how to integrate custom functionality (in this case, for logging) into asio's io_service, and how to use a custom service with basic_stream_socket<>.

- boost_asio/example/services/basic_logger.hpp
- boost_asio/example/services/daytime_client.cpp
- boost_asio/example/services/logger.hpp
- boost_asio/example/services/logger_service.cpp
- boost_asio/example/services/logger_service.hpp
- boost_asio/example/services/stream_socket_service.hpp

SOCKS 4

Example client program implementing the SOCKS 4 protocol for communication via a proxy.

- boost_asio/example/socks4/sync_client.cpp
- boost_asio/example/socks4/socks4.hpp

SSL

Example client and server programs showing the use of the ssl::stream<> template with asynchronous operations.

- boost_asio/example/ssl/client.cpp
- boost_asio/example/ssl/server.cpp

Timeouts

A collection of examples showing how to cancel long running asynchronous operations after a period of time.

- boost_asio/example/timeouts/async_tcp_client.cpp
- boost_asio/example/timeouts/blocking_tcp_client.cpp

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- boost_asio/example/timeouts/blocking_udp_client.cpp
- boost_asio/example/timeouts/server.cpp

Timers

Examples showing how to customise deadline_timer using different time types.

- boost_asio/example/timers/tick_count_timer.cpp
- boost_asio/example/timers/time_t_timer.cpp

Porthopper

Example illustrating mixed synchronous and asynchronous operations, and how to use Boost.Lambda with Boost.Asio.

- boost_asio/example/porthopper/protocol.hpp
- boost_asio/example/porthopper/client.cpp
- boost_asio/example/porthopper/server.cpp

Nonblocking

Example demonstrating reactor-style operations for integrating a third-party library that wants to perform the I/O operations itself.

- boost_asio/example/nonblocking/third_party_lib.cpp

UNIX Domain Sockets

Examples showing how to use UNIX domain (local) sockets.

- boost_asio/example/local/connect_pair.cpp
- boost_asio/example/local/stream_server.cpp
- boost_asio/example/local/stream_client.cpp

Windows

An example showing how to use the Windows-specific function `TransmitFile` with Boost.Asio.

- boost_asio/example/windows/transmit_file.cpp

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Reference

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Core

Classes

const_buffer
const_buffers_1
invalid_service_owner
io_service
io_service::id
io_service::service
io_service::strand
io_service::work
mutable_buffer
mutable_buffers_1
null_buffers
service_already_exists
streambuf

Class Templates

basic_io_object
basic_streambuf
buffered_read_stream
buffered_stream
buffered_write_stream
buffers_iterator

Free Functions

add_service
asio_handler_allocate
asio_handler_deallocate
asio_handler_invoke
async_read
async_read_at
async_read_until
async_write
async_write_at
buffer
buffer_cast
buffer_copy
buffer_size
buffers_begin
buffers_end
has_service
read
read_at
read_until
transfer_all
transfer_at_least
transfer_exactly
use_service
write
write_at

Placeholders

placeholders::bytes_transferred
placeholders::error
placeholders::iterator
placeholders::signal_number

Error Codes

error::basic_errors
error::netdb_errors
error::addrinfo_errors
error::misc_errors

Type Traits

is_match_condition
is_read_buffered
is_write_buffered

Type Requirements

Asynchronous operations
AsyncRandomAccessReadDevice
AsyncRandomAccessWriteDevice
AsyncReadStream
AsyncWriteStream
CompletionHandler
ConstBufferSequence
ConvertibleToConstBuffer
ConvertibleToMutableBuffer
Handler
IoObjectService
MutableBufferSequence
ReadHandler
Service
SyncRandomAccessReadDevice
SyncRandomAccessWriteDevice
SyncReadStream
SyncWriteStream
WriteHandler

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Networking

Classes

ip::address
ip::address_v4
ip::address_v6
ip::icmp
ip::icmp::endpoint
ip::icmp::resolver
ip::icmp::socket
ip::resolver_query_base
ip::tcp
ip::tcp::acceptor
ip::tcp::endpoint
ip::tcp::iostream
ip::tcp::resolver
ip::tcp::socket
ip::udp
ip::udp::endpoint
ip::udp::resolver
ip::udp::socket
socket_base

Free Functions

async_connect
connect
ip::host_name

Class Templates

basic_datagram_socket
basic_deadline_timer
basic_raw_socket
basic_seq_packet_socket
basic_socket
basic_socket_acceptor
basic_socket_iostream
basic_socket_streambuf
basic_stream_socket
ip::basic_endpoint
ip::basic_resolver
ip::basic_resolver_entry
ip::basic_resolver_iterator
ip::basic_resolver_query

Services

datagram_socket_service
ip::resolver_service
raw_socket_service
seq_packet_socket_service
socket_acceptor_service
stream_socket_service

Socket Options

ip::multicast::enable_loopback
ip::multicast::hops
ip::multicast::join_group
ip::multicast::leave_group
ip::multicast::outbound_interface
ip::tcp::no_delay
ip::unicast::hops
ip::v6_only
socket_base::broadcast
socket_base::debug
socket_base::do_not_route
socket_base::enable_connection_aborted
socket_base::keep_alive
socket_base::linger
socket_base::receive_buffer_size
socket_base::receive_low_watermark
socket_base::reuse_address
socket_base::send_buffer_size
socket_base::send_low_watermark

I/O Control Commands

socket_base::bytes_readable
socket_base::non_blocking_io

Type Requirements

AcceptHandler
ComposedConnectHandler
ConnectHandler
DatagramSocketService
Endpoint
GettableSocketOption
InternetProtocol
IoControlCommand
Protocol
RawSocketService
ResolveHandler
ResolverService
SeqPacketSocketService
SettableSocketOption
SocketAcceptorService
SocketService
StreamSocketService

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Timers

Classes

deadline_timer
high_resolution_timer
steady_timer
system_timer

Class Templates

basic_deadline_timer
basic_waitable_timer
time_traits
wait_traits

Services

deadline_timer_service
waitable_timer_service

Type Requirements

TimerService
TimeTraits
WaitableTimerService
WaitHandler
WaitTraits

SSL

Classes

ssl::context
ssl::context_base
ssl::rfc2818_verification
ssl::stream_base
ssl::verify_context

Class Templates

ssl::stream

Type Requirements

HandshakeHandler
ShutdownHandler

Serial Ports

Classes

serial_port
serial_port_base

Class Templates

basic_serial_port

Services

serial_port_service

Serial Port Options

serial_port_base::baud_rate
serial_port_base::flow_control
serial_port_base::parity
serial_port_base::stop_bits
serial_port_base::character_size

Type Requirements

GettableSerialPortOption
SerialPortService
SettableSerialPortOption

Signal Handling

Classes

signal_set

Class Templates

basic_signal_set

Services

signal_set_service

Type Requirements

SignalSetService
SignalHandler

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

POSIX-specific

Classes

local::stream_protocol
local::stream_protocol::acceptor
local::stream_protocol::endpoint
local::stream_protocol::iostream
local::stream_protocol::socket
local::datagram_protocol
local::datagram_protocol::endpoint
local::datagram_protocol::socket
posix::descriptor_base
posix::stream_descriptor

Free Functions

local::connect_pair

Class Templates

local::basic_endpoint
posix::basic_descriptor
posix::basic_stream_descriptor

Services

posix::stream_descriptor_service

Type Requirements

DescriptorService
StreamDescriptorService

Windows-specific

Classes

windows::object_handle
windows::overlapped_ptr
windows::random_access_handle
windows::stream_handle

Class Templates

windows::basic_handle
windows::basic_object_handle
windows::basic_random_access_handle
windows::basic_stream_handle

Services

windows::object_handle_service
windows::random_access_handle_service
windows::stream_handle_service

Type Requirements

HandleService
ObjectHandleService
RandomAccessHandleService
StreamHandleService

Requirements on asynchronous operations

In Boost.Asio, an asynchronous operation is initiated by a function that is named with the prefix `async_`. These functions will be referred to as *initiating functions*.

All initiating functions in Boost.Asio take a function object meeting handler requirements as the final parameter. These handlers accept as their first parameter an lvalue of type `const error_code`.

Implementations of asynchronous operations in Boost.Asio may call the application programming interface (API) provided by the operating system. If such an operating system API call results in an error, the handler will be invoked with a `const error_code` lvalue that evaluates to true. Otherwise the handler will be invoked with a `const error_code` lvalue that evaluates to false.

Unless otherwise noted, when the behaviour of an asynchronous operation is defined "as if" implemented by a *POSIX* function, the handler will be invoked with a value of type `error_code` that corresponds to the failure condition described by *POSIX* for that function, if any. Otherwise the handler will be invoked with an implementation-defined `error_code` value that reflects the operating system error.

Asynchronous operations will not fail with an error condition that indicates interruption by a signal (*POSIX* `EINTR`). Asynchronous operations will not fail with any error condition associated with non-blocking operations (*POSIX* `EWOULDBLOCK`, `EAGAIN` or `EINPROGRESS`; *Windows* `WSAEWOULDBLOCK` or `WSAEINPROGRESS`).

All asynchronous operations have an associated `io_service` object. Where the initiating function is a member function, the associated `io_service` is that returned by the `get_io_service()` member function on the same object. Where the initiating function is not a member function, the associated `io_service` is that returned by the `get_io_service()` member function of the first argument to the initiating function.

Arguments to initiating functions will be treated as follows:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

— If the parameter is declared as a const reference or by-value, the program is not required to guarantee the validity of the argument after the initiating function completes. The implementation may make copies of the argument, and all copies will be destroyed no later than immediately after invocation of the handler.

— If the parameter is declared as a non-const reference, const pointer or non-const pointer, the program must guarantee the validity of the argument until the handler is invoked.

The library implementation is only permitted to make calls to an initiating function's arguments' copy constructors or destructors from a thread that satisfies one of the following conditions:

— The thread is executing any member function of the associated `io_service` object.

— The thread is executing the destructor of the associated `io_service` object.

— The thread is executing one of the `io_service` service access functions `use_service`, `add_service` or `has_service`, where the first argument is the associated `io_service` object.

— The thread is executing any member function, constructor or destructor of an object of a class defined in this clause, where the object's `get_io_service()` member function returns the associated `io_service` object.

— The thread is executing any function defined in this clause, where any argument to the function has an `get_io_service()` member function that returns the associated `io_service` object.

> Boost.Asio may use one or more hidden threads to emulate asynchronous functionality. The above requirements are intended to prevent these hidden threads from making calls to program code. This means that a program can, for example, use thread-unsafe reference counting in handler objects, provided the program ensures that all calls to an `io_service` and related objects occur from the one thread.

The `io_service` object associated with an asynchronous operation will have unfinished work, as if by maintaining the existence of one or more objects of class `io_service::work` constructed using the `io_service`, until immediately after the handler for the asynchronous operation has been invoked.

When an asynchronous operation is complete, the handler for the operation will be invoked as if by:

1. Constructing a bound completion handler `bch` for the handler, as described below.

2. Calling `ios.post(bch)` to schedule the handler for deferred invocation, where `ios` is the associated `io_service`.

This implies that the handler must not be called directly from within the initiating function, even if the asynchronous operation completes immediately.

A bound completion handler is a handler object that contains a copy of a user-supplied handler, where the user-supplied handler accepts one or more arguments. The bound completion handler does not accept any arguments, and contains values to be passed as arguments to the user-supplied handler. The bound completion handler forwards the `asio_handler_allocate()`, `asio_handler_deallocate()`, and `asio_handler_invoke()` calls to the corresponding functions for the user-supplied handler. A bound completion handler meets the requirements for a completion handler.

For example, a bound completion handler for a `ReadHandler` may be implemented as follows:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<class ReadHandler>
struct bound_read_handler
{
  bound_read_handler(ReadHandler handler, const error_code& ec, size_t s)
    : handler_(handler), ec_(ec), s_(s)
  {
  }

  void operator()()
  {
    handler_(ec_, s_);
  }

  ReadHandler handler_;
  const error_code ec_;
  const size_t s_;
};

template<class ReadHandler>
void* asio_handler_allocate(size_t size,
                            bound_read_handler<ReadHandler>* this_handler)
{
  using boost::asio::asio_handler_allocate;
  return asio_handler_allocate(size, &this_handler->handler_);
}

template<class ReadHandler>
void asio_handler_deallocate(void* pointer, std::size_t size,
                             bound_read_handler<ReadHandler>* this_handler)
{
  using boost::asio::asio_handler_deallocate;
  asio_handler_deallocate(pointer, size, &this_handler->handler_);
}

template<class F, class ReadHandler>
void asio_handler_invoke(const F& f,
                         bound_read_handler<ReadHandler>* this_handler)
{
  using boost::asio::asio_handler_invoke;
  asio_handler_invoke(f, &this_handler->handler_);
}
```

If the thread that initiates an asynchronous operation terminates before the associated handler is invoked, the behaviour is implementation-defined. Specifically, on *Windows* versions prior to Vista, unfinished operations are cancelled when the initiating thread exits.

The handler argument to an initiating function defines a handler identity. That is, the original handler argument and any copies of the handler argument will be considered equivalent. If the implementation needs to allocate storage for an asynchronous operation, the implementation will perform `asio_handler_allocate(size, &h)`, where `size` is the required size in bytes, and `h` is the handler. The implementation will perform `asio_handler_deallocate(p, size, &h)`, where `p` is a pointer to the storage, to deallocate the storage prior to the invocation of the handler via `asio_handler_invoke`. Multiple storage blocks may be allocated for a single asynchronous operation.

Accept handler requirements

An accept handler must meet the requirements for a handler. A value `h` of an accept handler class should work correctly in the expression `h(ec)`, where `ec` is an lvalue of type `const error_code`.

Examples

A free function as an accept handler:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void accept_handler(
    const boost::system::error_code& ec)
{
  ...
}
```

An accept handler function object:

```
struct accept_handler
{
  ...
  void operator()(
      const boost::system::error_code& ec)
  {
    ...
  }
  ...
};
```

A non-static class member function adapted to an accept handler using `bind()`:

```
void my_class::accept_handler(
    const boost::system::error_code& ec)
{
  ...
}
...
acceptor.async_accept(...,
    boost::bind(&my_class::accept_handler,
      this, boost::asio::placeholders::error));
```

Buffer-oriented asynchronous random-access read device requirements

In the table below, `a` denotes an asynchronous random access read device object, `o` denotes an offset of type `boost::uint64_t`, `mb` denotes an object satisfying mutable buffer sequence requirements, and `h` denotes an object satisfying read handler requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 1. Buffer-oriented asynchronous random-access read device requirements

operation	type	semantics, pre/post-conditions
`a.get_io_service();`	`io_service&`	Returns the `io_service` object through which the `async_read_some_at` handler `h` will be invoked.
`a.async_read_some_at(o, mb, h);`	`void`	Initiates an asynchronous operation to read one or more bytes of data from the device `a` at the offset `o`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The mutable buffer sequence `mb` specifies memory where the data should be placed. The `async_read_some_at` operation shall always fill a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `mb` until such time as the read operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `mb` is destroyed, or — the handler for the asynchronous read operation is invoked, whichever comes first. If the total size of all buffers in the sequence `mb` is `0`, the asynchronous read operation shall complete immediately and pass `0` as the argument to the handler that specifies the number of bytes read.

Buffer-oriented asynchronous random-access write device requirements

In the table below, `a` denotes an asynchronous write stream object, `o` denotes an offset of type `boost::uint64_t`, `cb` denotes an object satisfying constant buffer sequence requirements, and `h` denotes an object satisfying write handler requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 2. Buffer-oriented asynchronous random-access write device requirements

operation	type	semantics, pre/post-conditions
`a.get_io_service();`	`io_service&`	Returns the `io_service` object through which the `async_write_some_at` handler `h` will be invoked.
`a.async_write_some_at(o, cb, h);`	`void`	Initiates an asynchronous operation to write one or more bytes of data to the device `a` at offset `o`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The constant buffer sequence `cb` specifies memory where the data to be written is located. The `async_write_some_at` operation shall always write a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `cb` until such time as the write operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `cb` is destroyed, or — the handler for the asynchronous write operation is invoked, whichever comes first. If the total size of all buffers in the sequence `cb` is `0`, the asynchronous write operation shall complete immediately and pass `0` as the argument to the handler that specifies the number of bytes written.

Buffer-oriented asynchronous read stream requirements

In the table below, `a` denotes an asynchronous read stream object, `mb` denotes an object satisfying mutable buffer sequence requirements, and `h` denotes an object satisfying read handler requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 3. Buffer-oriented asynchronous read stream requirements

operation	type	semantics, pre/post-conditions
`a.get_io_service();`	`io_service&`	Returns the `io_service` object through which the `async_read_some` handler `h` will be invoked.
`a.async_read_some(mb, h);`	`void`	Initiates an asynchronous operation to read one or more bytes of data from the stream `a`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The mutable buffer sequence `mb` specifies memory where the data should be placed. The `async_read_some` operation shall always fill a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `mb` until such time as the read operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `mb` is destroyed, or — the handler for the asynchronous read operation is invoked, whichever comes first. If the total size of all buffers in the sequence `mb` is `0`, the asynchronous read operation shall complete immediately and pass `0` as the argument to the handler that specifies the number of bytes read.

Buffer-oriented asynchronous write stream requirements

In the table below, `a` denotes an asynchronous write stream object, `cb` denotes an object satisfying constant buffer sequence requirements, and `h` denotes an object satisfying write handler requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 4. Buffer-oriented asynchronous write stream requirements

operation	type	semantics, pre/post-conditions
`a.get_io_service();`	`io_service&`	Returns the `io_service` object through which the `async_write_some` handler `h` will be invoked.
`a.async_write_some(cb, h);`	`void`	Initiates an asynchronous operation to write one or more bytes of data to the stream `a`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The constant buffer sequence `cb` specifies memory where the data to be written is located. The `async_write_some` operation shall always write a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `cb` until such time as the write operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `cb` is destroyed, or — the handler for the asynchronous write operation is invoked, whichever comes first. If the total size of all buffers in the sequence `cb` is `0`, the asynchronous write operation shall complete immediately and pass `0` as the argument to the handler that specifies the number of bytes written.

Completion handler requirements

A completion handler must meet the requirements for a handler. A value `h` of a completion handler class should work correctly in the expression `h()`.

Examples

A free function as a completion handler:

```
void completion_handler()
{
  ...
}
```

A completion handler function object:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
struct completion_handler
{
  ...
  void operator()()
  {
    ...
  }
  ...
};
```

A non-static class member function adapted to a completion handler using `bind()`:

```
void my_class::completion_handler()
{
  ...
}
...
my_io_service.post(boost::bind(&my_class::completion_handler, this));
```

Composed connect handler requirements

A composed connect handler must meet the requirements for a handler. A value `h` of a composed connect handler class should work correctly in the expression `h(ec, i)`, where `ec` is an lvalue of type `const error_code` and `i` is an lvalue of the type `Iterator` used in the corresponding `connect()` or async_connect()` function.

Examples

A free function as a composed connect handler:

```
void connect_handler(
    const boost::system::error_code& ec,
    boost::asio::ip::tcp::resolver::iterator iterator)
{
  ...
}
```

A composed connect handler function object:

```
struct connect_handler
{
  ...
  template <typename Iterator>
  void operator()(
      const boost::system::error_code& ec,
      Iterator iterator)
  {
    ...
  }
  ...
};
```

A non-static class member function adapted to a composed connect handler using `bind()`:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void my_class::connect_handler(
    const boost::system::error_code& ec,
    boost::asio::ip::tcp::resolver::iterator iterator)
{
  ...
}
...
boost::asio::async_connect(...,
    boost::bind(&my_class::connect_handler,
      this, boost::asio::placeholders::error,
      boost::asio::placeholders::iterator));
```

Connect handler requirements

A connect handler must meet the requirements for a handler. A value `h` of a connect handler class should work correctly in the expression `h(ec)`, where `ec` is an lvalue of type `const error_code`.

Examples

A free function as a connect handler:

```
void connect_handler(
    const boost::system::error_code& ec)
{
  ...
}
```

A connect handler function object:

```
struct connect_handler
{
  ...
  void operator()(
      const boost::system::error_code& ec)
  {
    ...
  }
  ...
};
```

A non-static class member function adapted to a connect handler using `bind()`:

```
void my_class::connect_handler(
    const boost::system::error_code& ec)
{
  ...
}
...
socket.async_connect(...,
    boost::bind(&my_class::connect_handler,
      this, boost::asio::placeholders::error));
```

Constant buffer sequence requirements

In the table below, `X` denotes a class containing objects of type `T`, `a` denotes a value of type `X` and `u` denotes an identifier.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 5. ConstBufferSequence requirements

expression	return type	assertion/note pre/post-condition
`X::value_type`	`T`	`T` meets the requirements for ConvertibleToConstBuffer.
`X::const_iterator`	iterator type pointing to `T`	`const_iterator` meets the requirements for bidirectional iterators (C++ Std, 24.1.4).

`X(a);`		post: `equal_const_buffer_seq(a, X(a))` where the binary predicate `equal_const_buffer_seq` is defined as
		`bool equal_const_buffer_seq(`
		`  const X& x1, const X& x2)`
		`{`
		`  return`
		`  distance(x1.be↵`
		`gin(), x1.end())`
		`  == distance(x2.be↵`
		`gin(), x2.end())`
		`  && equal(x1.be↵`
		`gin(), x1.end(),`
		`  x2.be↵`
		`gin(), equal_buffer);`
		`}`
		and the binary predicate `equal_buffer` is defined as
		`bool equal_buffer(`
		`  const X::value_type& v1,`
		`  const X::value_type& v2)`
		`{`
		`  const_buffer b1(v1);`
		`  const_buffer b2(v2);`
		`  return`
		`  buf↵`
		`fer_cast<const void*>(b1)`
		`  == buf↵`
		`fer_cast<const void*>(b2)`
		`  && buf↵`
		`fer_size(b1) == buf↵`
		`fer_size(b2);`
		`}`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition

`X u(a);`		post:
		`distance(a.begin(), a.end())`
		`  == distance(u.be↵`
		`gin(), u.end())`
		`  && equal(a.be↵`
		`gin(), a.end(),`
		`  u.be↵`
		`gin(), equal_buffer)`
		where the binary predicate `equal_buffer` is defined as
		`bool equal_buffer(`
		`  const X::value_type& v1,`
		`  const X::value_type& v2)`
		`{`
		`  const_buffer b1(v1);`
		`  const_buffer b2(v2);`
		`  return`
		`  buf↵`
		`fer_cast<const void*>(b1)`
		`  == buf↵`
		`fer_cast<const void*>(b2)`
		`  && buf↵`
		`fer_size(b1) == buf↵`
		`fer_size(b2);`
		`}`

`(&a)->~X();`	`void`	note: the destructor is applied to every element of `a`; all the memory is deallocated.
`a.begin();`	`const_iterator` or convertible to `const_iterator`	
`a.end();`	`const_iterator` or convertible to `const_iterator`	

Convertible to const buffer requirements

A type that meets the requirements for convertibility to a const buffer must meet the requirements of `CopyConstructible` types (C++ Std, 20.1.3), and the requirements of `Assignable` types (C++ Std, 23.1).

In the table below, `X` denotes a class meeting the requirements for convertibility to a const buffer, `a` and `b` denote values of type `X`, and `u`, `v` and `w` denote identifiers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 6. ConvertibleToConstBuffer requirements

expression	postcondition

`const_buffer u(a);`
`const_buffer v(a);`
	`buffer_cast<const void*>(u) == buf↵`
	`fer_cast<const void*>(v)`
	`  && buffer_size(u) == buffer_size(v)`

`const_buffer u(a);`
`const_buffer v = a;`
	`buffer_cast<const void*>(u) == buf↵`
	`fer_cast<const void*>(v)`
	`  && buffer_size(u) == buffer_size(v)`

`const_buffer u(a);`
`const_buffer v; v = a;`
	`buffer_cast<const void*>(u) == buf↵`
	`fer_cast<const void*>(v)`
	`  && buffer_size(u) == buffer_size(v)`

`const_buffer u(a);`
`const X& v = a;`
`const_buffer w(v);`
	`buffer_cast<const void*>(u) == buf↵`
	`fer_cast<const void*>(w)`
	`  && buffer_size(u) == buffer_size(w)`

`const_buffer u(a);`
`X v(a);`
`const_buffer w(v);`
	`buffer_cast<const void*>(u) == buf↵`
	`fer_cast<const void*>(w)`
	`  && buffer_size(u) == buffer_size(w)`

`const_buffer u(a);`
`X v = a;`
`const_buffer w(v);`
	`buffer_cast<const void*>(u) == buf↵`
	`fer_cast<const void*>(w)`
	`  && buffer_size(u) == buffer_size(w)`

`const_buffer u(a);`
`X v(b); v = a;`
`const_buffer w(v);`
	`buffer_cast<const void*>(u) == buf↵`
	`fer_cast<const void*>(w)`
	`  && buffer_size(u) == buffer_size(w)`

Convertible to mutable buffer requirements

A type that meets the requirements for convertibility to a mutable buffer must meet the requirements of `CopyConstructible` types (C++ Std, 20.1.3), and the requirements of `Assignable` types (C++ Std, 23.1).

In the table below, `X` denotes a class meeting the requirements for convertibility to a mutable buffer, `a` and `b` denote values of type `X`, and `u`, `v` and `w` denote identifiers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 7. ConvertibleToMutableBuffer requirements

expression	postcondition
`mutable_buffer u(a);` `mutable_buffer v(a);`	`buffer_cast<void*>(u) == buffer_cast<void*>(v)` `&& buffer_size(u) == buffer_size(v)`
`mutable_buffer u(a);` `mutable_buffer v = a;`	`buffer_cast<void*>(u) == buffer_cast<void*>(v)` `&& buffer_size(u) == buffer_size(v)`
`mutable_buffer u(a);` `mutable_buffer v; v = a;`	`buffer_cast<void*>(u) == buffer_cast<void*>(v)` `&& buffer_size(u) == buffer_size(v)`

`mutable_buffer u(a);`
`const X& v = a;`
`mutable_buffer w(v);`
	`buffer_cast<void*>(u) == buffer_cast<void*>(w)`
	`&& buffer_size(u) == buffer_size(w)`

`mutable_buffer u(a);`
`X v(a);`
`mutable_buffer w(v);`
	`buffer_cast<void*>(u) == buffer_cast<void*>(w)`
	`&& buffer_size(u) == buffer_size(w)`

`mutable_buffer u(a);`
`X v = a;`
`mutable_buffer w(v);`
	`buffer_cast<void*>(u) == buffer_cast<void*>(w)`
	`&& buffer_size(u) == buffer_size(w)`

`mutable_buffer u(a);`
`X v(b); v = a;`
`mutable_buffer w(v);`
	`buffer_cast<void*>(u) == buffer_cast<void*>(w)`
	`&& buffer_size(u) == buffer_size(w)`

Datagram socket service requirements

A datagram socket service must meet the requirements for a socket service, as well as the additional requirements listed below.

In the table below, `X` denotes a datagram socket service class for protocol `Protocol`, `a` denotes a value of type `X`, `b` denotes a value of type `X::implementation_type`, `e` denotes a value of type `Protocol::endpoint`, `ec` denotes a value of type `error_code`, `f` denotes a value of type `socket_base::message_flags`, `mb` denotes a value satisfying mutable buffer sequence requirements, `rh` denotes a value meeting `ReadHandler` requirements, `cb` denotes a value satisfying constant buffer sequence requirements, and `wh` denotes a value meeting `WriteHandler` requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 8. DatagramSocketService requirements

expression	return type	assertion/note pre/post-condition
`a.receive(b, mb, f, ec);`	`size_t`	pre: `a.is_open(b)`. Reads one or more bytes of data from a connected socket `b`. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes read. Otherwise returns `0`.
`a.async_receive(b, mb, f, rh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to read one or more bytes of data from a connected socket `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `mb` until such time as the read operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `mb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the operation completes successfully, the `ReadHandler` object `rh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.
`a.receive_from(b, mb, e, f, ec);`	`size_t`	pre: `a.is_open(b)`. Reads one or more bytes of data from an unconnected socket `b`. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes read. Otherwise returns `0`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.async_receive_from(b, mb, e, f, rh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to read one or more bytes of data from an unconnected socket `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `mb` until such time as the read operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `mb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. The program must ensure the object `e` is valid until the handler for the asynchronous operation is invoked. If the operation completes successfully, the `ReadHandler` object `rh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.
`a.send(b, cb, f, ec);`	`size_t`	pre: `a.is_open(b)`. Writes one or more bytes of data to a connected socket `b`. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes written. Otherwise returns `0`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.async_send(b, cb, f, wh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to write one or more bytes of data to a connected socket `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `cb` until such time as the write operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `cb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the operation completes successfully, the `WriteHandler` object `wh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.
`const typename Protocol::endpoint& u = e;` `a.send_to(b, cb, u, f, ec);`	`size_t`	pre: `a.is_open(b)`. Writes one or more bytes of data to an unconnected socket `b`. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes written. Otherwise returns `0`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`const typename Protocol::endpoint& u = e;` `a.async_send(b, cb, u, f, wh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to write one or more bytes of data to an unconnected socket `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `cb` until such time as the write operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `cb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the operation completes successfully, the `WriteHandler` object `wh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.

Descriptor service requirements

A descriptor service must meet the requirements for an I/O object service with support for movability, as well as the additional requirements listed below.

In the table below, `X` denotes a descriptor service class, `a` and `ao` denote values of type `X`, `b` and `c` denote values of type `X::implementation_type`, `n` denotes a value of type `X::native_handle_type`, `ec` denotes a value of type `error_code`, `i` denotes a value meeting `IoControlCommand` requirements, and `u` and `v` denote identifiers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 9. DescriptorService requirements

expression	return type	assertion/note pre/post-condition
`X::native_handle_type`		The implementation-defined native representation of a descriptor. Must satisfy the requirements of `CopyConstructible` types (C++ Std, 20.1.3), and the requirements of `Assignable` types (C++ Std, 23.1).
`a.construct(b);`		From IoObjectService requirements. post: `!a.is_open(b)`.
`a.destroy(b);`		From IoObjectService requirements. Implicitly cancels asynchronous operations, as if by calling `a.close(b, ec)`.
`a.move_construct(b, c);`		From IoObjectService requirements. The underlying native representation is moved from `c` to `b`.
`a.move_assign(b, ao, c);`		From IoObjectService requirements. Implicitly cancels asynchronous operations associated with `b`, as if by calling `a.close(b, ec)`. Then the underlying native representation is moved from `c` to `b`.
`a.assign(b, n, ec);`	`error_code`	pre: `!a.is_open(b)`. post: `!!ec || a.is_open(b)`.
`a.is_open(b);`	`bool`	

`const X& u = a;`
`const X::implementa↵`
`tion_type& v = b;`
`u.is_open(v);`
	`bool`	

`a.close(b, ec);`	`error_code`	If `a.is_open()` is true, causes any outstanding asynchronous operations to complete as soon as possible. Handlers for cancelled operations shall be passed the error code `error::operation_aborted`. post: `!a.is_open(b)`.
`a.native_handle(b);`	`X::native_handle_type`	

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.cancel(b, ec);`	`error_code`	pre: `a.is_open(b)`. Causes any outstanding asynchronous operations to complete as soon as possible. Handlers for cancelled operations shall be passed the error code `error::operation_aborted`.
`a.io_control(b, i, ec);`	`error_code`	pre: `a.is_open(b)`.

Endpoint requirements

An endpoint must meet the requirements of `CopyConstructible` types (C++ Std, 20.1.3), and the requirements of `Assignable` types (C++ Std, 23.1).

In the table below, `X` denotes an endpoint class, `a` denotes a value of type `X`, `s` denotes a size in bytes, and `u` denotes an identifier.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 10. Endpoint requirements

expression	type	assertion/note pre/post-conditions
`X::protocol_type`	type meeting protocol requirements	
`X u;`		
`X();`		
`a.protocol();`	`protocol_type`	
`a.data();`	a pointer	Returns a pointer suitable for passing as the *address* argument to *POSIX* functions such as `accept()`, `getpeername()`, `getsockname()` and `recvfrom()`. The implementation shall perform a `reinterpret_cast` on the pointer to convert it to `sockaddr*`.
`const X& u = a; u.data();`	a pointer	Returns a pointer suitable for passing as the *address* argument to *POSIX* functions such as `connect()`, or as the *dest_addr* argument to *POSIX* functions such as `sendto()`. The implementation shall perform a `reinterpret_cast` on the pointer to convert it to `const sockaddr*`.
`a.size();`	`size_t`	Returns a value suitable for passing as the *address_len* argument to *POSIX* functions such as `connect()`, or as the *dest_len* argument to *POSIX* functions such as `sendto()`, after appropriate integer conversion has been performed.
`a.resize(s);`		post: `a.size() == s` Passed the value contained in the *address_len* argument to *POSIX* functions such as `accept()`, `getpeername()`, `getsockname()` and `recvfrom()`, after successful completion of the function. Permitted to throw an exception if the protocol associated with the endpoint object `a` does not support the specified size.
`a.capacity();`	`size_t`	Returns a value suitable for passing as the *address_len* argument to *POSIX* functions such as `accept()`, `getpeername()`, `getsockname()` and `recvfrom()`, after appropriate integer conversion has been performed.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Gettable serial port option requirements

In the table below, `X` denotes a serial port option class, `a` denotes a value of `X`, `ec` denotes a value of type `error_code`, and `s` denotes a value of implementation-defined type *`storage`* (where *`storage`* is the type `DCB` on Windows and `termios` on *POSIX* platforms), and `u` denotes an identifier.

Table 11. GettableSerialPortOption requirements

expression	type	assertion/note pre/post-conditions
`const storage& u = s;` `a.load(u, ec);`	`error_code`	Retrieves the value of the serial port option from the storage. If successful, sets `ec` such that `!ec` is true. If an error occurred, sets `ec` such that `!!ec` is true. Returns `ec`.

Gettable socket option requirements

In the table below, `X` denotes a socket option class, `a` denotes a value of `X`, `p` denotes a value that meets the protocol requirements, and `u` denotes an identifier.

Table 12. GettableSocketOption requirements

expression	type	assertion/note pre/post-conditions
`a.level(p);`	`int`	Returns a value suitable for passing as the *level* argument to *POSIX* `getsockopt()` (or equivalent).
`a.name(p);`	`int`	Returns a value suitable for passing as the *option_name* argument to *POSIX* `getsockopt()` (or equivalent).
`a.data(p);`	a pointer, convertible to `void*`	Returns a pointer suitable for passing as the *option_value* argument to *POSIX* `getsockopt()` (or equivalent).
`a.size(p);`	`size_t`	Returns a value suitable for passing as the *option_len* argument to *POSIX* `getsockopt()` (or equivalent), after appropriate integer conversion has been performed.
`a.resize(p, s);`		post: `a.size(p) == s`. Passed the value contained in the *option_len* argument to *POSIX* `getsockopt()` (or equivalent) after successful completion of the function. Permitted to throw an exception if the socket option object `a` does not support the specified size.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Handlers

A handler must meet the requirements of `CopyConstructible` types (C++ Std, 20.1.3).

In the table below, `X` denotes a handler class, `h` denotes a value of `X`, `p` denotes a pointer to a block of allocated memory of type `void*`, `s` denotes the size for a block of allocated memory, and `f` denotes a function object taking no arguments.

Table 13. Handler requirements

expression	return type	assertion/note pre/post-conditions
`using boost::asio::asio_handler_allocate;` `asio_handler_allocate(s, &h);`	`void*`	Returns a pointer to a block of memory of size `s`. The pointer must satisfy the same alignment requirements as a pointer returned by `::operator new()`. Throws `bad_alloc` on failure. The `asio_handler_allocate()` function is located using argument-dependent lookup. The function `boost::asio::asio_handler_allocate()` serves as a default if no user-supplied function is available.

`using boost::asio::asio_handler_deallocate;`
`asio_handler_dealloc-`
`ate(p, s, &h);`
		Frees a block of memory associated with a pointer `p`, of at least size `s`, that was previously allocated using `asio_handler_allocate()`.
		The `asio_handler_deallocate()` function is located using argument-dependent lookup. The function `boost::asio::asio_handler_deallocate()` serves as a default if no user-supplied function is available.

`using boost::asio::asio_handler_invoke;` `asio_handler_invoke(f, &h);`		Causes the function object `f` to be executed as if by calling `f()`. The `asio_handler_invoke()` function is located using argument-dependent lookup. The function `boost::asio::asio_handler_invoke()` serves as a default if no user-supplied function is available.

Handle service requirements

A handle service must meet the requirements for an I/O object service with support for movability, as well as the additional requirements listed below.

In the table below, `X` denotes a handle service class, `a` and `ao` denote values of type `X`, `b` and `c` denote values of type `X::implementation_type`, `n` denotes a value of type `X::native_handle_type`, `ec` denotes a value of type `error_code`, and `u` and `v` denote identifiers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 14. HandleService requirements

expression	return type	assertion/note pre/post-condition
`X::native_handle_type`		The implementation-defined native representation of a handle. Must satisfy the requirements of `CopyConstructible` types (C++ Std, 20.1.3), and the requirements of `Assignable` types (C++ Std, 23.1).
`a.construct(b);`		From IoObjectService requirements. post: `!a.is_open(b)`.
`a.destroy(b);`		From IoObjectService requirements. Implicitly cancels asynchronous operations, as if by calling `a.close(b, ec)`.
`a.move_construct(b, c);`		From IoObjectService requirements. The underlying native representation is moved from `c` to `b`.
`a.move_assign(b, ao, c);`		From IoObjectService requirements. Implicitly cancels asynchronous operations associated with `b`, as if by calling `a.close(b, ec)`. Then the underlying native representation is moved from `c` to `b`.
`a.assign(b, n, ec);`	`error_code`	pre: `!a.is_open(b)`. post: `!!ec || a.is_open(b)`.
`a.is_open(b);`	`bool`	

`const X& u = a;`
`const X::implementa↵`
`tion_type& v = b;`
`u.is_open(v);`
	`bool`	

`a.close(b, ec);`	`error_code`	If `a.is_open()` is true, causes any outstanding asynchronous operations to complete as soon as possible. Handlers for cancelled operations shall be passed the error code `error::operation_aborted`. post: `!a.is_open(b)`.
`a.native_handle(b);`	`X::native_handle_type`	

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.cancel(b, ec);`	`error_code`	pre: `a.is_open(b)`. Causes any outstanding asynchronous operations to complete as soon as possible. Handlers for cancelled operations shall be passed the error code `error::operation_aborted`.

SSL handshake handler requirements

A handshake handler must meet the requirements for a handler. A value `h` of a handshake handler class should work correctly in the expression `h(ec)`, where `ec` is an lvalue of type `const error_code`.

Examples

A free function as a handshake handler:

```
void handshake_handler(
    const boost::system::error_code& ec)
{
  ...
}
```

A handshake handler function object:

```
struct handshake_handler
{
  ...
  void operator()(
      const boost::system::error_code& ec)
  {
    ...
  }
  ...
};
```

A non-static class member function adapted to a handshake handler using `bind()`:

```
void my_class::handshake_handler(
    const boost::system::error_code& ec)
{
  ...
}
...
ssl_stream.async_handshake(...,
    boost::bind(&my_class::handshake_handler,
      this, boost::asio::placeholders::error));
```

Internet protocol requirements

An internet protocol must meet the requirements for a protocol as well as the additional requirements listed below.

In the table below, `X` denotes an internet protocol class, `a` denotes a value of type `X`, and `b` denotes a value of type `X`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 15. InternetProtocol requirements

expression	return type	assertion/note pre/post-conditions
`X::resolver`	`ip::basic_resolver<X>`	The type of a resolver for the protocol.
`X::v4()`	`X`	Returns an object representing the IP version 4 protocol.
`X::v6()`	`X`	Returns an object representing the IP version 6 protocol.
`a == b`	convertible to `bool`	Returns whether two protocol objects are equal.
`a != b`	convertible to `bool`	Returns `!(a == b)`.

I/O control command requirements

In the table below, `X` denotes an I/O control command class, `a` denotes a value of `X`, and `u` denotes an identifier.

Table 16. IoControlCommand requirements

expression	type	assertion/note pre/post-conditions
`a.name();`	`int`	Returns a value suitable for passing as the *request* argument to *POSIX* `ioctl()` (or equivalent).
`a.data();`	a pointer, convertible to `void*`	

I/O object service requirements

An I/O object service must meet the requirements for a service, as well as the requirements listed below.

In the table below, `X` denotes an I/O object service class, `a` and `ao` denote values of type `X`, `b` and `c` denote values of type `X::implementation_type`, and `u` denotes an identifier.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 17. IoObjectService requirements

expression	return type	assertion/note pre/post-condition
`X::implementation_type`		
`X::implementation_type u;`		note: `X::implementation_type` has a public default constructor and destructor.
`a.construct(b);`		
`a.destroy(b);`		note: `destroy()` will only be called on a value that has previously been initialised with `construct()` or `move_construct()`.
`a.move_construct(b, c);`		note: only required for I/O objects that support movability.
`a.move_assign(b, ao, c);`		note: only required for I/O objects that support movability.

Mutable buffer sequence requirements

In the table below, `X` denotes a class containing objects of type `T`, `a` denotes a value of type `X` and `u` denotes an identifier.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 18. MutableBufferSequence requirements

expression	return type	assertion/note pre/post-condition
`X::value_type`	`T`	`T` meets the requirements for ConvertibleToMutableBuffer.
`X::const_iterator`	iterator type pointing to `T`	`const_iterator` meets the requirements for bidirectional iterators (C++ Std, 24.1.4).

`X(a);`		post: `equal_mutable_buffer_seq(a, X(a))` where the binary predicate `equal_mutable_buffer_seq` is defined as
		`bool equal_mutable_buf↵fer_seq(`
		`  const X& x1, const X& x2)`
		`{`
		`  return`
		`  distance(x1.be↵gin(), x1.end())`
		`  == distance(x2.be↵gin(), x2.end())`
		`  && equal(x1.be↵gin(), x1.end(),`
		`  x2.be↵gin(), equal_buffer);`
		`}`
		and the binary predicate `equal_buffer` is defined as
		`bool equal_buffer(`
		`  const X::value_type& v1,`
		`  const X::value_type& v2)`
		`{`
		`  mutable_buffer b1(v1);`
		`  mutable_buffer b2(v2);`
		`  return`
		`  buf↵fer_cast<const void*>(b1)`
		`  == buf↵fer_cast<const void*>(b2)`
		`  && buf↵fer_size(b1) == buf↵fer_size(b2);`
		`}`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition

`X u(a);`		post:
		`distance(a.begin(), a.end())`
		`  == distance(u.begin(), u.end())`
		`  && equal(a.begin(), a.end(),`
		`  u.begin(), equal_buffer)`
		where the binary predicate `equal_buffer` is defined as
		`bool equal_buffer(`
		`  const X::value_type& v1,`
		`  const X::value_type& v2)`
		`{`
		`  mutable_buffer b1(v1);`
		`  mutable_buffer b2(v2);`
		`  return`
		`  buf.`
		`buffer_cast<const void*>(b1)`
		`  == buf.`
		`buffer_cast<const void*>(b2)`
		`  && buf.`
		`buffer_size(b1) == buf.`
		`buffer_size(b2);`
		`}`

`(&a)->~X();`	`void`	note: the destructor is applied to every element of `a`; all the memory is deallocated.
`a.begin();`	`const_iterator` or convertible to `const_iterator`	
`a.end();`	`const_iterator` or convertible to `const_iterator`	

Object handle service requirements

An object handle service must meet the requirements for a handle service, as well as the additional requirements listed below.

In the table below, `X` denotes an object handle service class, `a` denotes a value of type `X`, `b` denotes a value of type `X::implementation_type`, `ec` denotes a value of type `error_code`, and `wh` denotes a value meeting `WaitHandler` requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 19. ObjectHandleService requirements

expression	return type	assertion/note pre/post-condition
`a.wait(b, ec);`	`error_code`	pre: `a.is_open(b)`. Synchronously waits for the object represented by handle `b` to become signalled.
`a.async_wait(b, wh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to wait for the object represented by handle `b` to become signalled. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements.

Protocol requirements

A protocol must meet the requirements of `CopyConstructible` types (C++ Std, 20.1.3), and the requirements of `Assignable` types (C++ Std, 23.1).

In the table below, `X` denotes a protocol class, and `a` denotes a value of `X`.

Table 20. Protocol requirements

expression	return type	assertion/note pre/post-conditions
`X::endpoint`	type meeting endpoint requirements	
`a.family()`	`int`	Returns a value suitable for passing as the *domain* argument to *POSIX* `socket()` (or equivalent).
`a.type()`	`int`	Returns a value suitable for passing as the *type* argument to *POSIX* `socket()` (or equivalent).
`a.protocol()`	`int`	Returns a value suitable for passing as the *protocol* argument to *POSIX* `socket()` (or equivalent).

Random access handle service requirements

A random access handle service must meet the requirements for a handle service, as well as the additional requirements listed below.

In the table below, `X` denotes a random access handle service class, `a` denotes a value of type `X`, `b` denotes a value of type `X::implementation_type`, `ec` denotes a value of type `error_code`, `o` denotes an offset of type boost::uint64_t, `mb` denotes a value satisfying mutable buffer sequence requirements, `rh` denotes a value meeting `ReadHandler` requirements, `cb` denotes a value satisfying constant buffer sequence requirements, and `wh` denotes a value meeting `WriteHandler` requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 21. RandomAccessHandleService requirements

expression	return type	assertion/note pre/post-condition
`a.read_some_at(b, o, mb, ec);`	`size_t`	pre: `a.is_open(b)`. Reads one or more bytes of data from a handle `b` at offset `o`. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes read. Otherwise returns `0`. If the total size of all buffers in the sequence `mb` is `0`, the function shall return `0` immediately.
`a.async_read_some_at(b, o, mb, rh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to read one or more bytes of data from a handle `b` at offset `o`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `mb` until such time as the read operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `mb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the total size of all buffers in the sequence `mb` is `0`, the asynchronous read operation shall complete immediately and pass `0` as the argument to the handler that specifies the number of bytes read. If the operation completes successfully, the `ReadHandler` object `rh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.write_some_at(b, o, cb, ec);`	`size_t`	pre: `a.is_open(b)`. Writes one or more bytes of data to a handle `b` at offset `o`. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes written. Otherwise returns `0`. If the total size of all buffers in the sequence `cb` is `0`, the function shall return `0` immediately.
`a.async_write_some_at(b, o, cb, wh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to write one or more bytes of data to a handle `b` at offset `o`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `cb` until such time as the write operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `cb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the total size of all buffers in the sequence `cb` is `0`, the asynchronous operation shall complete immediately and pass `0` as the argument to the handler that specifies the number of bytes read. If the operation completes successfully, the `WriteHandler` object `wh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.

Raw socket service requirements

A raw socket service must meet the requirements for a socket service, as well as the additional requirements listed below.

In the table below, `X` denotes a raw socket service class for protocol `Protocol`, `a` denotes a value of type `X`, `b` denotes a value of type `X::implementation_type`, `e` denotes a value of type `Protocol::endpoint`, `ec` denotes a value of type `error_code`, `f` denotes a value of type `socket_base::message_flags`, `mb` denotes a value satisfying mutable buffer sequence requirements, `rh`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

denotes a value meeting `ReadHandler` requirements, `cb` denotes a value satisfying constant buffer sequence requirements, and `wh` denotes a value meeting `WriteHandler` requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 22. RawSocketService requirements

expression	return type	assertion/note pre/post-condition
`a.receive(b, mb, f, ec);`	`size_t`	pre: `a.is_open(b)`. Reads one or more bytes of data from a connected socket `b`. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes read. Otherwise returns `0`.
`a.async_receive(b, mb, f, rh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to read one or more bytes of data from a connected socket `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `mb` until such time as the read operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `mb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the operation completes successfully, the `ReadHandler` object `rh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.
`a.receive_from(b, mb, e, f, ec);`	`size_t`	pre: `a.is_open(b)`. Reads one or more bytes of data from an unconnected socket `b`. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes read. Otherwise returns `0`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.async_receive_from(b, mb, e, f, rh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to read one or more bytes of data from an unconnected socket `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `mb` until such time as the read operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `mb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. The program must ensure the object `e` is valid until the handler for the asynchronous operation is invoked. If the operation completes successfully, the `ReadHandler` object `rh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.
`a.send(b, cb, f, ec);`	`size_t`	pre: `a.is_open(b)`. Writes one or more bytes of data to a connected socket `b`. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes written. Otherwise returns `0`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.async_send(b, cb, f, wh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to write one or more bytes of data to a connected socket `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `cb` until such time as the write operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `cb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the operation completes successfully, the `WriteHandler` object `wh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.
`const typename Protocol::endpoint& u = e;` `a.send_to(b, cb, u, f, ec);`	`size_t`	pre: `a.is_open(b)`. Writes one or more bytes of data to an unconnected socket `b`. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes written. Otherwise returns `0`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`const typename Protocol::endpoint& u = e;` `a.async_send(b, cb, u, f, wh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to write one or more bytes of data to an unconnected socket `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `cb` until such time as the write operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `cb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the operation completes successfully, the `WriteHandler` object `wh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.

Read handler requirements

A read handler must meet the requirements for a handler. A value `h` of a read handler class should work correctly in the expression `h(ec, s)`, where `ec` is an lvalue of type `const error_code` and `s` is an lvalue of type `const size_t`.

Examples

A free function as a read handler:

```
void read_handler(
    const boost::system::error_code& ec,
    std::size_t bytes_transferred)
{
  ...
}
```

A read handler function object:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
struct read_handler
{
  ...
  void operator()(
      const boost::system::error_code& ec,
      std::size_t bytes_transferred)
  {
    ...
  }
  ...
};
```

A non-static class member function adapted to a read handler using `bind()`:

```
void my_class::read_handler(
    const boost::system::error_code& ec,
    std::size_t bytes_transferred)
{
  ...
}
...
socket.async_read(...,
    boost::bind(&my_class::read_handler,
      this, boost::asio::placeholders::error,
      boost::asio::placeholders::bytes_transferred));
```

Resolve handler requirements

A resolve handler must meet the requirements for a handler. A value `h` of a resolve handler class should work correctly in the expression `h(ec, i)`, where `ec` is an lvalue of type `const error_code` and `i` is an lvalue of type `const ip::basic_resolver_iterator<InternetProtocol>`. `InternetProtocol` is the template parameter of the `resolver_service` which is used to initiate the asynchronous operation.

Examples

A free function as a resolve handler:

```
void resolve_handler(
    const boost::system::error_code& ec,
    boost::asio::ip::tcp::resolver::iterator iterator)
{
  ...
}
```

A resolve handler function object:

```
struct resolve_handler
{
  ...
  void operator()(
      const boost::system::error_code& ec,
      boost::asio::ip::tcp::resolver::iterator iterator)
  {
    ...
  }
  ...
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

A non-static class member function adapted to a resolve handler using `bind()`:

```
void my_class::resolve_handler(
    const boost::system::error_code& ec,
    boost::asio::ip::tcp::resolver::iterator iterator)
{
  ...
}
...
resolver.async_resolve(...,
    boost::bind(&my_class::resolve_handler,
      this, boost::asio::placeholders::error,
      boost::asio::placeholders::iterator));
```

Resolver service requirements

A resolver service must meet the requirements for an I/O object service, as well as the additional requirements listed below.

In the table below, `X` denotes a resolver service class for protocol `InternetProtocol`, `a` denotes a value of type `X`, `b` denotes a value of type `X::implementation_type`, `q` denotes a value of type `ip::basic_resolver_query<InternetProtocol>`, `e` denotes a value of type `ip::basic_endpoint<InternetProtocol>`, `ec` denotes a value of type `error_code`, and `h` denotes a value meeting `ResolveHandler` requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 23. ResolverService requirements

expression	return type	assertion/note pre/post-condition
`a.destroy(b);`		From IoObjectService requirements. Implicitly cancels asynchronous resolve operations, as if by calling `a.cancel(b, ec)`.
`a.cancel(b, ec);`	`error_code`	Causes any outstanding asynchronous resolve operations to complete as soon as possible. Handlers for cancelled operations shall be passed the error code `error::operation_aborted`.
`a.resolve(b, q, ec);`	`ip::basic_resolver_iterator<InternetProtocol>`	On success, returns an iterator `i` such that `i != ip::basic_resolver_iterator<InternetProtocol>()`. Otherwise returns `ip::basic_resolver_iterator<InternetProtocol>()`.
`a.async_resolve(b, q, h);`		Initiates an asynchronous resolve operation that is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. If the operation completes successfully, the `ResolveHandler` object `h` shall be invoked with an iterator object `i` such that the condition `i != ip::basic_resolver_iterator<InternetProtocol>()` holds. Otherwise it is invoked with `ip::basic_resolver_iterator<InternetProtocol>()`.
`a.resolve(b, e, ec);`	`ip::basic_resolver_iterator<InternetProtocol>`	On success, returns an iterator `i` such that `i != ip::basic_resolver_iterator<InternetProtocol>()`. Otherwise returns `ip::basic_resolver_iterator<InternetProtocol>()`.
`a.async_resolve(b, e, h);`		Initiates an asynchronous resolve operation that is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. If the operation completes successfully, the `ResolveHandler` object `h` shall be invoked with an iterator object `i` such that the condition `i != ip::basic_resolver_iterator<InternetProtocol>()` holds. Otherwise it is invoked with `ip::basic_resolver_iterator<InternetProtocol>()`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Sequenced packet socket service requirements

A sequenced packet socket service must meet the requirements for a socket service, as well as the additional requirements listed below.

In the table below, `X` denotes a stream socket service class, `a` denotes a value of type `X`, `b` denotes a value of type `X::implementation_type`, `ec` denotes a value of type `error_code`, `f` denotes a value of type `socket_base::message_flags`, `g` denotes an lvalue of type `socket_base::message_flags`, `mb` denotes a value satisfying mutable buffer sequence requirements, `rh` denotes a value meeting `ReadHandler` requirements, `cb` denotes a value satisfying constant buffer sequence requirements, and `wh` denotes a value meeting `WriteHandler` requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 24. StreamSocketService requirements

expression	return type	assertion/note pre/post-condition
`a.receive(b, mb, f, g, ec);`	`size_t`	pre: `a.is_open(b)`. Reads one or more bytes of data from a connected socket `b`. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. If successful, sets `g` to the flags associated with the received data, and returns the number of bytes read. Otherwise, sets `g` to `0` and returns `0`.
`a.async_receive(b, mb, f, g, rh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to read one or more bytes of data from a connected socket `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `mb` until such time as the read operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `mb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the operation completes successfully, sets `g` to the flags associated with the received data, then invokes the `ReadHandler` object `rh` with the number of bytes transferred. Otherwise, sets `g` to `0` and invokes `rh` with `0` bytes transferred.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.send(b, cb, f, ec);`	`size_t`	pre: `a.is_open(b)`. Writes one or more bytes of data to a connected socket `b`. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes written. Otherwise returns `0`.
`a.async_send(b, cb, f, wh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to write one or more bytes of data to a connected socket `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `cb` until such time as the write operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `cb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the operation completes successfully, the `WriteHandler` object `wh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.

Serial port service requirements

A serial port service must meet the requirements for an I/O object service with support for movability, as well as the additional requirements listed below.

In the table below, `X` denotes a serial port service class, `a` and `ao` denote values of type `X`, `d` denotes a serial port device name of type `std::string`, `b` and `c` denote values of type `X::implementation_type`, `n` denotes a value of type `X::native_handle_type`, `ec` denotes a value of type `error_code`, `s` denotes a value meeting `SettableSerialPortOption` requirements, `g` denotes a value meeting `GettableSerialPortOption` requirements, `mb` denotes a value satisfying mutable buffer sequence requirements, `rh` denotes a value meeting `ReadHandler` requirements, `cb` denotes a value satisfying constant buffer sequence requirements, and `wh` denotes a value meeting `WriteHandler` requirements. and `u` and `v` denote identifiers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 25. SerialPortService requirements

expression	return type	assertion/note pre/post-condition
`X::native_handle_type`		The implementation-defined native representation of a serial port. Must satisfy the requirements of `CopyConstructible` types (C++ Std, 20.1.3), and the requirements of `Assignable` types (C++ Std, 23.1).
`a.construct(b);`		From IoObjectService requirements. post: `!a.is_open(b)`.
`a.destroy(b);`		From IoObjectService requirements. Implicitly cancels asynchronous operations, as if by calling `a.close(b, ec)`.
`a.move_construct(b, c);`		From IoObjectService requirements. The underlying native representation is moved from `c` to `b`.
`a.move_assign(b, ao, c);`		From IoObjectService requirements. Implicitly cancels asynchronous operations associated with `b`, as if by calling `a.close(b, ec)`. Then the underlying native representation is moved from `c` to `b`.
`const std::string& u = d;` `a.open(b, u, ec);`	`error_code`	pre: `!a.is_open(b)`. post: `!!ec || a.is_open(b)`.
`a.assign(b, n, ec);`	`error_code`	pre: `!a.is_open(b)`. post: `!!ec || a.is_open(b)`.
`a.is_open(b);`	`bool`	

`const X& u = a;`
`const X::implementa↵`
`tion_type& v = b;`
`u.is_open(v);`
	`bool`	

`a.close(b, ec);`	`error_code`	If `a.is_open()` is true, causes any outstanding asynchronous operations to complete as soon as possible. Handlers for cancelled operations shall be passed the error code `error::operation_aborted`. post: `!a.is_open(b)`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.native_handle(b);`	`X::native_handle_type`	
`a.cancel(b, ec);`	`error_code`	pre: `a.is_open(b)`. Causes any outstanding asynchronous operations to complete as soon as possible. Handlers for cancelled operations shall be passed the error code `error::operation_aborted`.
`a.set_option(b, s, ec);`	`error_code`	pre: `a.is_open(b)`.
`a.get_option(b, g, ec);`	`error_code`	pre: `a.is_open(b)`.

`const X& u = a;`
`const X::implementa↵`
`tion_type& v = b;`
`u.get_option(v, g, ec);`
	`error_code`	pre: `a.is_open(b)`.

`a.send_break(b, ec);`	`error_code`	pre: `a.is_open(b)`.
`a.read_some(b, mb, ec);`	`size_t`	pre: `a.is_open(b)`. Reads one or more bytes of data from a serial port `b`. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes read. Otherwise returns `0`. If the total size of all buffers in the sequence `mb` is `0`, the function shall return `0` immediately. If the operation completes due to graceful connection closure by the peer, the operation shall fail with `error::eof`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.async_read_some(b, mb, rh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to read one or more bytes of data from a serial port `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `mb` until such time as the read operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `mb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the total size of all buffers in the sequence `mb` is `0`, the asynchronous read operation shall complete immediately and pass `0` as the argument to the handler that specifies the number of bytes read. If the operation completes due to graceful connection closure by the peer, the operation shall fail with `error::eof`. If the operation completes successfully, the `ReadHandler` object `rh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.
`a.write_some(b, cb, ec);`	`size_t`	pre: `a.is_open(b)`. Writes one or more bytes of data to a serial port `b`. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes written. Otherwise returns `0`. If the total size of all buffers in the sequence `cb` is `0`, the function shall return `0` immediately.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.async_write_some(b, cb, wh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to write one or more bytes of data to a serial port `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `cb` until such time as the write operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `cb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the total size of all buffers in the sequence `cb` is `0`, the asynchronous operation shall complete immediately and pass `0` as the argument to the handler that specifies the number of bytes read. If the operation completes successfully, the `WriteHandler` object `wh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.

Service requirements

A class is a service if it is publicly derived from another service, or if it is a class derived from `io_service::service` and contains a publicly-accessible declaration as follows:

```
static io_service::id id;
```

All services define a one-argument constructor that takes a reference to the `io_service` object that owns the service. This constructor is *explicit*, preventing its participation in automatic conversions. For example:

```
class my_service : public io_service::service
{
public:
  static io_service::id id;
  explicit my_service(io_service& ios);
private:
  virtual void shutdown_service();
  ...
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

A service's `shutdown_service` member function must cause all copies of user-defined handler objects that are held by the service to be destroyed.

Settable serial port option requirements

In the table below, `X` denotes a serial port option class, `a` denotes a value of `X`, `ec` denotes a value of type `error_code`, and `s` denotes a value of implementation-defined type *`storage`* (where *`storage`* is the type `DCB` on Windows and `termios` on *POSIX* platforms), and `u` denotes an identifier.

Table 26. SettableSerialPortOption requirements

expression	type	assertion/note pre/post-conditions
`const X& u = a;` `u.store(s, ec);`	`error_code`	Saves the value of the serial port option to the storage. If successful, sets `ec` such that `!ec` is true. If an error occurred, sets `ec` such that `!!ec` is true. Returns `ec`.

Settable socket option requirements

In the table below, `X` denotes a socket option class, `a` denotes a value of `X`, `p` denotes a value that meets the protocol requirements, and `u` denotes an identifier.

Table 27. SettableSocketOption requirements

expression	type	assertion/note pre/post-conditions
`a.level(p);`	`int`	Returns a value suitable for passing as the *level* argument to *POSIX* `setsockopt()` (or equivalent).
`a.name(p);`	`int`	Returns a value suitable for passing as the *option_name* argument to *POSIX* `setsockopt()` (or equivalent).
`const X& u = a; u.data(p);`	a pointer, convertible to `const void*`	Returns a pointer suitable for passing as the *option_value* argument to *POSIX* `setsockopt()` (or equivalent).
`a.size(p);`	`size_t`	Returns a value suitable for passing as the *option_len* argument to *POSIX* `setsockopt()` (or equivalent), after appropriate integer conversion has been performed.

SSL shutdown handler requirements

A shutdown handler must meet the requirements for a handler. A value `h` of a shutdown handler class should work correctly in the expression `h(ec)`, where `ec` is an lvalue of type `const error_code`.

Examples

A free function as a shutdown handler:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void shutdown_handler(
    const boost::system::error_code& ec)
{
  ...
}
```

A shutdown handler function object:

```
struct shutdown_handler
{
  ...
  void operator()(
      const boost::system::error_code& ec)
  {
    ...
  }
  ...
};
```

A non-static class member function adapted to a shutdown handler using `bind()`:

```
void my_class::shutdown_handler(
    const boost::system::error_code& ec)
{
  ...
}
...
ssl_stream.async_shutdown(
    boost::bind(&my_class::shutdown_handler,
      this, boost::asio::placeholders::error));
```

Signal handler requirements

A signal handler must meet the requirements for a handler. A value `h` of a signal handler class should work correctly in the expression `h(ec, n)`, where `ec` is an lvalue of type `const error_code` and `n` is an lvalue of type `const int`.

Examples

A free function as a signal handler:

```
void signal_handler(
    const boost::system::error_code& ec,
    int signal_number)
{
  ...
}
```

A signal handler function object:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
struct signal_handler
{
  ...
  void operator()(
      const boost::system::error_code& ec,
      int signal_number)
  {
    ...
  }
  ...
};
```

A non-static class member function adapted to a signal handler using `bind()`:

```
void my_class::signal_handler(
    const boost::system::error_code& ec,
    int signal_number)
{
  ...
}
...
my_signal_set.async_wait(
    boost::bind(&my_class::signal_handler,
      this, boost::asio::placeholders::error,
      boost::asio::placeholders::signal_number));
```

Signal set service requirements

A signal set service must meet the requirements for an I/O object service, as well as the additional requirements listed below.

In the table below, `X` denotes a signal set service class, `a` denotes a value of type `X`, `b` denotes a value of type `X::implementation_type`, `ec` denotes a value of type `error_code`, `n` denotes a value of type `int`, and `sh` denotes a value meeting `SignalHandler` requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 28. SignalSetService requirements

expression	return type	assertion/note pre/post-condition
`a.construct(b);`		From IoObjectService requirements.
`a.destroy(b);`		From IoObjectService requirements. Implicitly clears the registered signals as if by calling `a.clear(b, ec)`, then implicitly cancels outstanding asynchronous operations as if by calling `a.cancel(b, ec)`.
`a.add(b, n, ec);`	`error_code`	
`a.remove(b, n, ec);`	`error_code`	
`a.clear(b, ec);`	`error_code`	
`a.cancel(b, ec);`	`error_code`	
`a.async_wait(b, sh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to wait for the delivery of one of the signals registered for the signal set `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. If the operation completes successfully, the `SignalHandler` object `sh` is invoked with the number identifying the delivered signal. Otherwise it is invoked with `0`.

Socket acceptor service requirements

A socket acceptor service must meet the requirements for an I/O object service, as well as the additional requirements listed below.

In the table below, `X` denotes a socket acceptor service class for protocol `Protocol`, `a` and `ao` denote values of type `X`, `b` and `c` denote values of type `X::implementation_type`, `p` denotes a value of type `Protocol`, `n` denotes a value of type `X::native_handle_type`, `e` denotes a value of type `Protocol::endpoint`, `ec` denotes a value of type `error_code`, `s` denotes a value meeting `SettableSocketOption` requirements, `g` denotes a value meeting `GettableSocketOption` requirements, `i` denotes a value meeting `IoControlCommand` requirements, `k` denotes a value of type `basic_socket<Protocol, SocketService>` where `SocketService` is a type meeting socket service requirements, `ah` denotes a value meeting `AcceptHandler` requirements, and `u` and `v` denote identifiers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 29. SocketAcceptorService requirements

expression	return type	assertion/note pre/post-condition
`X::native_handle_type`		The implementation-defined native representation of a socket acceptor. Must satisfy the requirements of `CopyConstructible` types (C++ Std, 20.1.3), and the requirements of `Assignable` types (C++ Std, 23.1).
`a.construct(b);`		From IoObjectService requirements. post: `!a.is_open(b)`.
`a.destroy(b);`		From IoObjectService requirements. Implicitly cancels asynchronous operations, as if by calling `a.close(b, ec)`.
`a.move_construct(b, c);`		From IoObjectService requirements. The underlying native representation is moved from `c` to `b`.
`a.move_assign(b, ao, c);`		From IoObjectService requirements. Implicitly cancels asynchronous operations associated with `b`, as if by calling `a.close(b, ec)`. Then the underlying native representation is moved from `c` to `b`.
`a.open(b, p, ec);`	`error_code`	pre: `!a.is_open(b)`. post: `!!ec || a.is_open(b)`.
`a.assign(b, p, n, ec);`	`error_code`	pre: `!a.is_open(b)`. post: `!!ec || a.is_open(b)`.
`a.is_open(b);`	`bool`	

`const X& u = a;`
`const X::implementa↵`
`tion_type& v = b;`
`u.is_open(v);`
	`bool`	

`a.close(b, ec);`	`error_code`	If `a.is_open()` is true, causes any outstanding asynchronous operations to complete as soon as possible. Handlers for cancelled operations shall be passed the error code `error::operation_aborted`. post: `!a.is_open(b)`.
`a.native_handle(b);`	`X::native_handle_type`	

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.cancel(b, ec);`	`error_code`	pre: `a.is_open(b)`. Causes any outstanding asynchronous operations to complete as soon as possible. Handlers for cancelled operations shall be passed the error code `error::operation_aborted`.
`a.set_option(b, s, ec);`	`error_code`	pre: `a.is_open(b)`.
`a.get_option(b, g, ec);`	`error_code`	pre: `a.is_open(b)`.

`const X& u = a;`
`const X::implementa↵`
`tion_type& v = b;`
`u.get_option(v, g, ec);`
	`error_code`	pre: `a.is_open(b)`.

`a.io_control(b, i, ec);`	`error_code`	pre: `a.is_open(b)`.

`const typename Protocol::end↵`
`point& u = e;`
`a.bind(b, u, ec);`
	`error_code`	pre: `a.is_open(b)`.

`a.local_endpoint(b, ec);`	`Protocol::endpoint`	pre: `a.is_open(b)`.

`const X& u = a;`
`const X::implementa↵`
`tion_type& v = b;`
`u.local_endpoint(v, ec);`
	`Protocol::endpoint`	pre: `a.is_open(b)`.

`a.accept(b, k, &e, ec);`	`error_code`	pre: `a.is_open(b) && !k.is_open()`. post: `k.is_open()`
`a.accept(b, k, 0, ec);`	`error_code`	pre: `a.is_open(b) && !k.is_open()`. post: `k.is_open()`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.async_accept(b, k, &e, ah);`		`pre: a.is_open(b) && !k.is_open()`. Initiates an asynchronous accept operation that is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The program must ensure the objects `k` and `e` are valid until the handler for the asynchronous operation is invoked.
`a.async_accept(b, k, 0, ah);`		`pre: a.is_open(b) && !k.is_open()`. Initiates an asynchronous accept operation that is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The program must ensure the object `k` is valid until the handler for the asynchronous operation is invoked.

Socket service requirements

A socket service must meet the requirements for an I/O object service with support for movability, as well as the additional requirements listed below.

In the table below, `X` denotes a socket service class for protocol `Protocol`, `a` and `ao` denote values of type `X`, `b` and `c` denote values of type `X::implementation_type`, `p` denotes a value of type `Protocol`, `n` denotes a value of type `X::native_handle_type`, `e` denotes a value of type `Protocol::endpoint`, `ec` denotes a value of type `error_code`, `s` denotes a value meeting `SettableSocketOption` requirements, `g` denotes a value meeting `GettableSocketOption` requirements, `i` denotes a value meeting `IoControlCommand` requirements, `h` denotes a value of type `socket_base::shutdown_type`, `ch` denotes a value meeting `ConnectHandler` requirements, and `u` and `v` denote identifiers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 30. SocketService requirements

expression	return type	assertion/note pre/post-condition
`X::native_handle_type`		The implementation-defined native representation of a socket. Must satisfy the requirements of `CopyConstructible` types (C++ Std, 20.1.3), and the requirements of `Assignable` types (C++ Std, 23.1).
`a.construct(b);`		From IoObjectService requirements. post: `!a.is_open(b)`.
`a.destroy(b);`		From IoObjectService requirements. Implicitly cancels asynchronous operations, as if by calling `a.close(b, ec)`.
`a.move_construct(b, c);`		From IoObjectService requirements. The underlying native representation is moved from `c` to `b`.
`a.move_assign(b, ao, c);`		From IoObjectService requirements. Implicitly cancels asynchronous operations associated with `b`, as if by calling `a.close(b, ec)`. Then the underlying native representation is moved from `c` to `b`.
`a.open(b, p, ec);`	`error_code`	pre: `!a.is_open(b)`. post: `!!ec || a.is_open(b)`.
`a.assign(b, p, n, ec);`	`error_code`	pre: `!a.is_open(b)`. post: `!!ec || a.is_open(b)`.
`a.is_open(b);`	`bool`	

`const X& u = a;`
`const X::implementa↵`
`tion_type& v = b;`
`u.is_open(v);`
	`bool`	

`a.close(b, ec);`	`error_code`	If `a.is_open()` is true, causes any outstanding asynchronous operations to complete as soon as possible. Handlers for cancelled operations shall be passed the error code `error::operation_aborted`. post: `!a.is_open(b)`.
`a.native_handle(b);`	`X::native_handle_type`	

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.cancel(b, ec);`	`error_code`	pre: `a.is_open(b)`. Causes any outstanding asynchronous operations to complete as soon as possible. Handlers for cancelled operations shall be passed the error code `error::operation_aborted`.
`a.set_option(b, s, ec);`	`error_code`	pre: `a.is_open(b)`.
`a.get_option(b, g, ec);`	`error_code`	pre: `a.is_open(b)`.

`const X& u = a;`
`const X::implementa↵`
`tion_type& v = b;`
`u.get_option(v, g, ec);`
	`error_code`	pre: `a.is_open(b)`.

`a.io_control(b, i, ec);`	`error_code`	pre: `a.is_open(b)`.
`a.at_mark(b, ec);`	`bool`	pre: `a.is_open(b)`.

`const X& u = a;`
`const X::implementa↵`
`tion_type& v = b;`
`u.at_mark(v, ec);`
	`bool`	pre: `a.is_open(b)`.

`a.available(b, ec);`	`size_t`	pre: `a.is_open(b)`.

`const X& u = a;`
`const X::implementa↵`
`tion_type& v = b;`
`u.available(v, ec);`
	`size_t`	pre: `a.is_open(b)`.

`const typename Protocol::end↵`
`point& u = e;`
`a.bind(b, u, ec);`
	`error_code`	pre: `a.is_open(b)`.

`a.shutdown(b, h, ec);`	`error_code`	pre: `a.is_open(b)`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.local_endpoint(b, ec);`	`Protocol::endpoint`	pre: `a.is_open(b)`.

`const X& u = a;`
`const X::implementa↵`
`tion_type& v = b;`
`u.local_endpoint(v, ec);`
	`Protocol::endpoint`	pre: `a.is_open(b)`.

`a.remote_endpoint(b, ec);`	`Protocol::endpoint`	pre: `a.is_open(b)`.

`const X& u = a;`
`const X::implementa↵`
`tion_type& v = b;`
`u.remote_endpoint(v, ec);`
	`Protocol::endpoint`	pre: `a.is_open(b)`.

`const typename Protocol::end↵`
`point& u = e;`
`a.connect(b, u, ec);`
	`error_code`	pre: `a.is_open(b)`.

`const typename Protocol::end↵`
`point& u = e;`
`a.async_connect(b, u, ch);`
		pre: `a.is_open(b)`.
		Initiates an asynchronous connect operation that is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements.

Stream descriptor service requirements

A stream descriptor service must meet the requirements for a descriptor service, as well as the additional requirements listed below.

In the table below, `X` denotes a stream descriptor service class, `a` denotes a value of type `X`, `b` denotes a value of type `X::implementation_type`, `ec` denotes a value of type `error_code`, `mb` denotes a value satisfying mutable buffer sequence requirements, `rh` denotes a value meeting `ReadHandler` requirements, `cb` denotes a value satisfying constant buffer sequence requirements, and `wh` denotes a value meeting `WriteHandler` requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 31. StreamDescriptorService requirements

expression	return type	assertion/note pre/post-condition
`a.read_some(b, mb, ec);`	`size_t`	pre: `a.is_open(b)`. Reads one or more bytes of data from a descriptor `b`. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes read. Otherwise returns `0`. If the total size of all buffers in the sequence `mb` is `0`, the function shall return `0` immediately. If the operation completes due to graceful connection closure by the peer, the operation shall fail with `error::eof`.
`a.async_read_some(b, mb, rh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to read one or more bytes of data from a descriptor `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `mb` until such time as the read operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `mb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the total size of all buffers in the sequence `mb` is `0`, the asynchronous read operation shall complete immediately and pass `0` as the argument to the handler that specifies the number of bytes read. If the operation completes due to graceful connection closure by the peer, the operation shall fail with `error::eof`. If the operation completes successfully, the `ReadHandler` object `rh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.write_some(b, cb, ec);`	`size_t`	pre: `a.is_open(b)`. Writes one or more bytes of data to a descriptor `b`. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes written. Otherwise returns `0`. If the total size of all buffers in the sequence `cb` is `0`, the function shall return `0` immediately.
`a.async_write_some(b, cb, wh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to write one or more bytes of data to a descriptor `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `cb` until such time as the write operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `cb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the total size of all buffers in the sequence `cb` is `0`, the asynchronous operation shall complete immediately and pass `0` as the argument to the handler that specifies the number of bytes read. If the operation completes successfully, the `WriteHandler` object `wh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.

Stream handle service requirements

A stream handle service must meet the requirements for a handle service, as well as the additional requirements listed below.

In the table below, `X` denotes a stream handle service class, `a` denotes a value of type `X`, `b` denotes a value of type `X::implementation_type`, `ec` denotes a value of type `error_code`, `mb` denotes a value satisfying mutable buffer sequence requirements, `rh` denotes a value meeting `ReadHandler` requirements, `cb` denotes a value satisfying constant buffer sequence requirements, and `wh` denotes a value meeting `WriteHandler` requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 32. StreamHandleService requirements

expression	return type	assertion/note pre/post-condition
`a.read_some(b, mb, ec);`	`size_t`	pre: `a.is_open(b)`. Reads one or more bytes of data from a handle `b`. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes read. Otherwise returns `0`. If the total size of all buffers in the sequence `mb` is `0`, the function shall return `0` immediately. If the operation completes due to graceful connection closure by the peer, the operation shall fail with `error::eof`.
`a.async_read_some(b, mb, rh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to read one or more bytes of data from a handle `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `mb` until such time as the read operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `mb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the total size of all buffers in the sequence `mb` is `0`, the asynchronous read operation shall complete immediately and pass `0` as the argument to the handler that specifies the number of bytes read. If the operation completes due to graceful connection closure by the peer, the operation shall fail with `error::eof`. If the operation completes successfully, the `ReadHandler` object `rh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.write_some(b, cb, ec);`	`size_t`	pre: `a.is_open(b)`. Writes one or more bytes of data to a handle `b`. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes written. Otherwise returns `0`. If the total size of all buffers in the sequence `cb` is `0`, the function shall return `0` immediately.
`a.async_write_some(b, cb, wh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to write one or more bytes of data to a handle `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `cb` until such time as the write operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `cb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the total size of all buffers in the sequence `cb` is `0`, the asynchronous operation shall complete immediately and pass `0` as the argument to the handler that specifies the number of bytes read. If the operation completes successfully, the `WriteHandler` object `wh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.

Stream socket service requirements

A stream socket service must meet the requirements for a socket service, as well as the additional requirements listed below.

In the table below, `X` denotes a stream socket service class, `a` denotes a value of type `X`, `b` denotes a value of type `X::implementation_type`, `ec` denotes a value of type `error_code`, `f` denotes a value of type `socket_base::message_flags`, `mb` denotes a value satisfying mutable buffer sequence requirements, `rh` denotes a value meeting `ReadHandler` requirements, `cb` denotes a value satisfying constant buffer sequence requirements, and `wh` denotes a value meeting `WriteHandler` requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 33. StreamSocketService requirements

expression	return type	assertion/note pre/post-condition
`a.receive(b, mb, f, ec);`	`size_t`	pre: `a.is_open(b)`. Reads one or more bytes of data from a connected socket `b`. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes read. Otherwise returns `0`. If the total size of all buffers in the sequence `mb` is `0`, the function shall return `0` immediately. If the operation completes due to graceful connection closure by the peer, the operation shall fail with `error::eof`.
`a.async_receive(b, mb, f, rh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to read one or more bytes of data from a connected socket `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The mutable buffer sequence `mb` specifies memory where the data should be placed. The operation shall always fill a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `mb` until such time as the read operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `mb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the total size of all buffers in the sequence `mb` is `0`, the asynchronous read operation shall complete immediately and pass `0` as the argument to the handler that specifies the number of bytes read. If the operation completes due to graceful connection closure by the peer, the operation shall fail with `error::eof`. If the operation completes successfully, the `ReadHandler` object `rh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

expression	return type	assertion/note pre/post-condition
`a.send(b, cb, f, ec);`	`size_t`	pre: `a.is_open(b)`. Writes one or more bytes of data to a connected socket `b`. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes written. Otherwise returns `0`. If the total size of all buffers in the sequence `cb` is `0`, the function shall return `0` immediately.
`a.async_send(b, cb, f, wh);`	`void`	pre: `a.is_open(b)`. Initiates an asynchronous operation to write one or more bytes of data to a connected socket `b`. The operation is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The constant buffer sequence `cb` specifies memory where the data to be written is located. The operation shall always write a buffer in the sequence completely before proceeding to the next. The implementation shall maintain one or more copies of `cb` until such time as the write operation no longer requires access to the memory specified by the buffers in the sequence. The program must ensure the memory is valid until: — the last copy of `cb` is destroyed, or — the handler for the asynchronous operation is invoked, whichever comes first. If the total size of all buffers in the sequence `cb` is `0`, the asynchronous operation shall complete immediately and pass `0` as the argument to the handler that specifies the number of bytes read. If the operation completes successfully, the `WriteHandler` object `wh` is invoked with the number of bytes transferred. Otherwise it is invoked with `0`.

Buffer-oriented synchronous random-access read device requirements

In the table below, `a` denotes a synchronous random-access read device object, `o` denotes an offset of type `boost::uint64_t`, `mb` denotes an object satisfying mutable buffer sequence requirements, and `ec` denotes an object of type `error_code`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 34. Buffer-oriented synchronous random-access read device requirements

operation	type	semantics, pre/post-conditions

`a.read_some_at(o, mb);`	`size_t`	Equivalent to:
		`error_code ec;`
		`size_t s = a.read_some_at(o, mb, ec);`
		`if (ec) throw system_er↵`
		`ror(ec);`
		`return s;`

`a.read_some_at(o, mb, ec);`	`size_t`	Reads one or more bytes of data from the device `a` at offset `o`. The mutable buffer sequence `mb` specifies memory where the data should be placed. The `read_some_at` operation shall always fill a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes read and sets `ec` such that `!ec` is true. If an error occurred, returns `0` and sets `ec` such that `!!ec` is true. If the total size of all buffers in the sequence `mb` is `0`, the function shall return `0` immediately.

Buffer-oriented synchronous random-access write device requirements

In the table below, `a` denotes a synchronous random-access write device object, `o` denotes an offset of type `boost::uint64_t`, `cb` denotes an object satisfying constant buffer sequence requirements, and `ec` denotes an object of type `error_code`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 35. Buffer-oriented synchronous random-access write device requirements

operation	type	semantics, pre/post-conditions

`a.write_some_at(o, cb);`	`size_t`	Equivalent to:
		`error_code ec;`
		`size_t s = a.write_some(o, cb, ec);`
		`if (ec) throw system_er↵`
		`ror(ec);`
		`return s;`

`a.write_some_at(o, cb, ec);`	`size_t`	Writes one or more bytes of data to the device `a` at offset `o`. The constant buffer sequence `cb` specifies memory where the data to be written is located. The `write_some_at` operation shall always write a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes written and sets `ec` such that `!ec` is true. If an error occurred, returns `0` and sets `ec` such that `!!ec` is true. If the total size of all buffers in the sequence `cb` is `0`, the function shall return `0` immediately.

Buffer-oriented synchronous read stream requirements

In the table below, `a` denotes a synchronous read stream object, `mb` denotes an object satisfying mutable buffer sequence requirements, and `ec` denotes an object of type `error_code`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 36. Buffer-oriented synchronous read stream requirements

operation	type	semantics, pre/post-conditions

`a.read_some(mb);`	`size_t`	Equivalent to:
		`error_code ec;`
		`size_t s = a.read_some(mb, ec);`
		`if (ec) throw system_er↵`
		`ror(ec);`
		`return s;`

`a.read_some(mb, ec);`	`size_t`	Reads one or more bytes of data from the stream `a`. The mutable buffer sequence `mb` specifies memory where the data should be placed. The `read_some` operation shall always fill a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes read and sets `ec` such that `!ec` is true. If an error occurred, returns `0` and sets `ec` such that `!!ec` is true. If the total size of all buffers in the sequence `mb` is `0`, the function shall return `0` immediately.

Buffer-oriented synchronous write stream requirements

In the table below, `a` denotes a synchronous write stream object, `cb` denotes an object satisfying constant buffer sequence requirements, and `ec` denotes an object of type `error_code`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 37. Buffer-oriented synchronous write stream requirements

operation	type	semantics, pre/post-conditions

`a.write_some(cb);`	`size_t`	Equivalent to:
		`error_code ec;`
		`size_t s = a.write_some(cb, ec);`
		`if (ec) throw system_er↵`
		`ror(ec);`
		`return s;`

`a.write_some(cb, ec);`	`size_t`	Writes one or more bytes of data to the stream `a`. The constant buffer sequence `cb` specifies memory where the data to be written is located. The `write_some` operation shall always write a buffer in the sequence completely before proceeding to the next. If successful, returns the number of bytes written and sets `ec` such that `!ec` is true. If an error occurred, returns `0` and sets `ec` such that `!!ec` is true. If the total size of all buffers in the sequence `cb` is `0`, the function shall return `0` immediately.

Time traits requirements

In the table below, `X` denotes a time traits class for time type `Time`, `t`, `t1`, and `t2` denote values of type `Time`, and `d` denotes a value of type `X::duration_type`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 38. TimeTraits requirements

expression	return type	assertion/note pre/post-condition
`X::time_type`	`Time`	Represents an absolute time. Must support default construction, and meet the requirements for `CopyConstructible` and `Assignable`.
`X::duration_type`		Represents the difference between two absolute times. Must support default construction, and meet the requirements for `CopyConstructible` and `Assignable`. A duration can be positive, negative, or zero.
`X::now();`	`time_type`	Returns the current time.
`X::add(t, d);`	`time_type`	Returns a new absolute time resulting from adding the duration `d` to the absolute time `t`.
`X::subtract(t1, t2);`	`duration_type`	Returns the duration resulting from subtracting `t2` from `t1`.
`X::less_than(t1, t2);`	`bool`	Returns whether `t1` is to be treated as less than `t2`.
`X::to_posix_duration(d);`	`date_time::time_duration_type`	Returns the `date_time::time_duration_type` value that most closely represents the duration `d`.

Timer service requirements

A timer service must meet the requirements for an I/O object service, as well as the additional requirements listed below.

In the table below, `X` denotes a timer service class for time type `Time` and traits type `TimeTraits`, `a` denotes a value of type `X`, `b` denotes a value of type `X::implementation_type`, `t` denotes a value of type `Time`, `d` denotes a value of type `TimeTraits::duration_type`, `e` denotes a value of type `error_code`, and `h` denotes a value meeting `WaitHandler` requirements.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Table 39. TimerService requirements

expression	return type	assertion/note pre/post-condition
`a.destroy(b);`		From IoObjectService requirements. Implicitly cancels asynchronous wait operations, as if by calling `a.cancel(b, e)`.
`a.cancel(b, e);`	`size_t`	Causes any outstanding asynchronous wait operations to complete as soon as possible. Handlers for cancelled operations shall be passed the error code `error::operation_aborted`. Sets `e` to indicate success or failure. Returns the number of operations that were cancelled.
`a.expires_at(b);`	`Time`	
`a.expires_at(b, t, e);`	`size_t`	Implicitly cancels asynchronous wait operations, as if by calling `a.cancel(b, e)`. Returns the number of operations that were cancelled. post: `a.expires_at(b) == t`.
`a.expires_from_now(b);`	`TimeTraits::duration_type`	Returns a value equivalent to `TimeTraits::subtract(a.expires_at(b), TimeTraits::now())`.
`a.expires_from_now(b, d, e);`	`size_t`	Equivalent to `a.expires_at(b, TimeTraits::add(TimeTraits::now(), d), e)`.
`a.wait(b, e);`	`error_code`	Sets `e` to indicate success or failure. Returns `e`. `post: !!e || !TimeTraits::lt(TimeTraits::now(), a.expires_at(b))`.
`a.async_wait(b, h);`		Initiates an asynchronous wait operation that is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The handler shall be posted for execution only if the condition `!!ec || !TimeTraits::lt(TimeTraits::now(), a.expires_at(b))` holds, where `ec` is the error code to be passed to the handler.

Waitable timer service requirements

A waitable timer service must meet the requirements for an I/O object service, as well as the additional requirements listed below.

In the table below, `X` denotes a waitable timer service class for clock type `Clock`, where `Clock` meets the C++11 clock type requirements, `a` denotes a value of type `X`, `b` denotes a value of type `X::implementation_type`, `t` denotes a value of type

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

`Clock::time_point`, `d` denotes a value of type `Clock::duration`, `e` denotes a value of type `error_code`, and `h` denotes a value meeting `WaitHandler` requirements.

Table 40. WaitableTimerService requirements

expression	return type	assertion/note pre/post-condition
`a.destroy(b);`		From IoObjectService requirements. Implicitly cancels asynchronous wait operations, as if by calling `a.cancel(b, e)`.
`a.cancel(b, e);`	`size_t`	Causes any outstanding asynchronous wait operations to complete as soon as possible. Handlers for cancelled operations shall be passed the error code `error::operation_aborted`. Sets `e` to indicate success or failure. Returns the number of operations that were cancelled.
`a.expires_at(b);`	`Clock::time_point`	
`a.expires_at(b, t, e);`	`size_t`	Implicitly cancels asynchronous wait operations, as if by calling `a.cancel(b, e)`. Returns the number of operations that were cancelled. post: `a.expires_at(b) == t`.
`a.expires_from_now(b);`	`Clock::duration`	Returns a value equivalent to `a.expires_at(b) - Clock::now()`.
`a.expires_from_now(b, d, e);`	`size_t`	Equivalent to `a.expires_at(b, Clock::now() + d, e)`.
`a.wait(b, e);`	`error_code`	Sets `e` to indicate success or failure. Returns `e`. post: `!!e || !(Clock::now() < a.expires_at(b))`.
`a.async_wait(b, h);`		Initiates an asynchronous wait operation that is performed via the `io_service` object `a.get_io_service()` and behaves according to asynchronous operation requirements. The handler shall be posted for execution only if the condition `!!ec || !(Clock::now() < a.expires_at(b))` holds, where `ec` is the error code to be passed to the handler.

Wait handler requirements

A wait handler must meet the requirements for a handler. A value `h` of a wait handler class should work correctly in the expression `h(ec)`, where `ec` is an lvalue of type `const error_code`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Examples

A free function as a wait handler:

```
void wait_handler(
    const boost::system::error_code& ec)
{
  ...
}
```

A wait handler function object:

```
struct wait_handler
{
  ...
  void operator()(
      const boost::system::error_code& ec)
  {
    ...
  }
  ...
};
```

A non-static class member function adapted to a wait handler using `bind()`:

```
void my_class::wait_handler(
    const boost::system::error_code& ec)
{
  ...
}
...
socket.async_wait(...,
    boost::bind(&my_class::wait_handler,
      this, boost::asio::placeholders::error));
```

Wait traits requirements

In the table below, `X` denotes a wait traits class for clock type `Clock`, where `Clock` meets the C++11 type requirements for a clock, and `d` denotes a value of type `Clock::duration`.

Table 41. WaitTraits requirements

expression	return type	assertion/note pre/post-condition
`X::to_wait_duration(d);`	`Clock::duration`	Returns the maximum duration to be used for an individual, implementation-defined wait operation.

Write handler requirements

A write handler must meet the requirements for a handler. A value `h` of a write handler class should work correctly in the expression `h(ec, s)`, where `ec` is an lvalue of type `const error_code` and `s` is an lvalue of type `const size_t`.

Examples

A free function as a write handler:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void write_handler(
    const boost::system::error_code& ec,
    std::size_t bytes_transferred)
{
  ...
}
```

A write handler function object:

```
struct write_handler
{
  ...
  void operator()(
      const boost::system::error_code& ec,
      std::size_t bytes_transferred)
  {
    ...
  }
  ...
};
```

A non-static class member function adapted to a write handler using `bind()`:

```
void my_class::write_handler(
    const boost::system::error_code& ec,
    std::size_t bytes_transferred)
{
  ...
}
...
socket.async_write(...,
    boost::bind(&my_class::write_handler,
      this, boost::asio::placeholders::error,
      boost::asio::placeholders::bytes_transferred));
```

add_service

```
template<
    typename Service>
void add_service(
    io_service & ios,
    Service * svc);
```

This function is used to add a service to the `io_service`.

Parameters

ios The `io_service` object that owns the service.

svc The service object. On success, ownership of the service object is transferred to the `io_service`. When the `io_service` object is destroyed, it will destroy the service object by performing:

```
delete static_cast<io_service::service*>(svc)
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::asio::service_already_exists	Thrown if a service of the given type is already present in the `io_service`.
boost::asio::invalid_service_owner	Thrown if the service's owning `io_service` is not the `io_service` object specified by the ios parameter.

Requirements

Header: `boost/asio/io_service.hpp`

Convenience header: `boost/asio.hpp`

asio_handler_allocate

Default allocation function for handlers.

```
void * asio_handler_allocate(
    std::size_t size,
    ... );
```

Asynchronous operations may need to allocate temporary objects. Since asynchronous operations have a handler function object, these temporary objects can be said to be associated with the handler.

Implement asio_handler_allocate and asio_handler_deallocate for your own handlers to provide custom allocation for these temporary objects.

This default implementation is simply:

```
return ::operator new(size);
```

Remarks

All temporary objects associated with a handler will be deallocated before the upcall to the handler is performed. This allows the same memory to be reused for a subsequent asynchronous operation initiated by the handler.

Example

```
class my_handler;

void* asio_handler_allocate(std::size_t size, my_handler* context)
{
  return ::operator new(size);
}

void asio_handler_deallocate(void* pointer, std::size_t size,
    my_handler* context)
{
  ::operator delete(pointer);
}
```

Requirements

Header: `boost/asio/handler_alloc_hook.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

asio_handler_deallocate

Default deallocation function for handlers.

```
void asio_handler_deallocate(
    void * pointer,
    std::size_t size,
    ... );
```

Implement asio_handler_allocate and asio_handler_deallocate for your own handlers to provide custom allocation for the associated temporary objects.

This default implementation is simply:

```
::operator delete(pointer);
```

Requirements

Header: `boost/asio/handler_alloc_hook.hpp`

Convenience header: `boost/asio.hpp`

asio_handler_invoke

Default invoke function for handlers.

```
template<
    typename Function>
void asio_handler_invoke(
    Function function,
    ... );
```

Completion handlers for asynchronous operations are invoked by the `io_service` associated with the corresponding object (e.g. a socket or deadline_timer). Certain guarantees are made on when the handler may be invoked, in particular that a handler can only be invoked from a thread that is currently calling `run()` on the corresponding `io_service` object. Handlers may subsequently be invoked through other objects (such as `io_service::strand` objects) that provide additional guarantees.

When asynchronous operations are composed from other asynchronous operations, all intermediate handlers should be invoked using the same method as the final handler. This is required to ensure that user-defined objects are not accessed in a way that may violate the guarantees. This hooking function ensures that the invoked method used for the final handler is accessible at each intermediate step.

Implement asio_handler_invoke for your own handlers to specify a custom invocation strategy.

This default implementation is simply:

```
function();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
class my_handler;

template <typename Function>
void asio_handler_invoke(Function function, my_handler* context)
{
  context->strand_.dispatch(function);
}
```

Requirements

Header: `boost/asio/handler_invoke_hook.hpp`

Convenience header: `boost/asio.hpp`

async_connect

Asynchronously establishes a socket connection by trying each endpoint in a sequence.

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ComposedConnectHandler>
void async_connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    ComposedConnectHandler handler);
  » more...

template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ComposedConnectHandler>
void async_connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    Iterator end,
    ComposedConnectHandler handler);
  » more...

template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ConnectCondition,
    typename ComposedConnectHandler>
void async_connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    ConnectCondition connect_condition,
    ComposedConnectHandler handler);
  » more...

template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ConnectCondition,
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    typename ComposedConnectHandler>
void async_connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    Iterator end,
    ConnectCondition connect_condition,
    ComposedConnectHandler handler);
  » more...
```

Requirements

Header: `boost/asio/connect.hpp`

Convenience header: `boost/asio.hpp`

async_connect (1 of 4 overloads)

Asynchronously establishes a socket connection by trying each endpoint in a sequence.

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ComposedConnectHandler>
void async_connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    ComposedConnectHandler handler);
```

This function attempts to connect a socket to one of a sequence of endpoints. It does this by repeated calls to the socket's `async_connect` member function, once for each endpoint in the sequence, until a connection is successfully established.

Parameters

s — The socket to be connected. If the socket is already open, it will be closed.

begin — An iterator pointing to the start of a sequence of endpoints.

handler — The handler to be called when the connect operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  // Result of operation. if the sequence is empty, set to
  // boost::asio::error::not_found. Otherwise, contains the
  // error from the last connection attempt.
  const boost::system::error_code& error,

  // On success, an iterator denoting the successfully
  // connected endpoint. Otherwise, the end iterator.
  Iterator iterator
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

This overload assumes that a default constructed object of type `Iterator` represents the end of the sequence. This is a valid assumption for iterator types such as `boost::asio::ip::tcp::resolver::iterator`.

Example

```
tcp::resolver r(io_service);
tcp::resolver::query q("host", "service");
tcp::socket s(io_service);

// ...

r.async_resolve(q, resolve_handler);

// ...

void resolve_handler(
    const boost::system::error_code& ec,
    tcp::resolver::iterator i)
{
  if (!ec)
  {
    boost::asio::async_connect(s, i, connect_handler);
  }
}

// ...

void connect_handler(
    const boost::system::error_code& ec,
    tcp::resolver::iterator i)
{
  // ...
}
```

async_connect (2 of 4 overloads)

Asynchronously establishes a socket connection by trying each endpoint in a sequence.

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ComposedConnectHandler>
void async_connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    Iterator end,
    ComposedConnectHandler handler);
```

This function attempts to connect a socket to one of a sequence of endpoints. It does this by repeated calls to the socket's `async_connect` member function, once for each endpoint in the sequence, until a connection is successfully established.

Parameters

s	The socket to be connected. If the socket is already open, it will be closed.
begin	An iterator pointing to the start of a sequence of endpoints.
end	An iterator pointing to the end of a sequence of endpoints.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

handler The handler to be called when the connect operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  // Result of operation. if the sequence is empty, set to
  // boost::asio::error::not_found. Otherwise, contains the
  // error from the last connection attempt.
  const boost::system::error_code& error,

  // On success, an iterator denoting the successfully
  // connected endpoint. Otherwise, the end iterator.
  Iterator iterator
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

```
tcp::resolver r(io_service);
tcp::resolver::query q("host", "service");
tcp::socket s(io_service);

// ...

r.async_resolve(q, resolve_handler);

// ...

void resolve_handler(
    const boost::system::error_code& ec,
    tcp::resolver::iterator i)
{
  if (!ec)
  {
    tcp::resolver::iterator end;
    boost::asio::async_connect(s, i, end, connect_handler);
  }
}

// ...

void connect_handler(
    const boost::system::error_code& ec,
    tcp::resolver::iterator i)
{
  // ...
}
```

async_connect (3 of 4 overloads)

Asynchronously establishes a socket connection by trying each endpoint in a sequence.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ConnectCondition,
    typename ComposedConnectHandler>
void async_connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    ConnectCondition connect_condition,
    ComposedConnectHandler handler);
```

This function attempts to connect a socket to one of a sequence of endpoints. It does this by repeated calls to the socket's `async_connect` member function, once for each endpoint in the sequence, until a connection is successfully established.

Parameters

s — The socket to be connected. If the socket is already open, it will be closed.

begin — An iterator pointing to the start of a sequence of endpoints.

connect_condition — A function object that is called prior to each connection attempt. The signature of the function object must be:

```
Iterator connect_condition(
    const boost::system::error_code& ec,
    Iterator next);
```

The `ec` parameter contains the result from the most recent connect operation. Before the first connection attempt, `ec` is always set to indicate success. The `next` parameter is an iterator pointing to the next endpoint to be tried. The function object should return the next iterator, but is permitted to return a different iterator so that endpoints may be skipped. The implementation guarantees that the function object will never be called with the end iterator.

handler — The handler to be called when the connect operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  // Result of operation. if the sequence is empty, set to
  // boost::asio::error::not_found. Otherwise, contains the
  // error from the last connection attempt.
  const boost::system::error_code& error,

  // On success, an iterator denoting the successfully
  // connected endpoint. Otherwise, the end iterator.
  Iterator iterator
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

This overload assumes that a default constructed object of type `Iterator` represents the end of the sequence. This is a valid assumption for iterator types such as `boost::asio::ip::tcp::resolver::iterator`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

The following connect condition function object can be used to output information about the individual connection attempts:

```
struct my_connect_condition
{
  template <typename Iterator>
  Iterator operator()(
      const boost::system::error_code& ec,
      Iterator next)
  {
    if (ec) std::cout << "Error: " << ec.message() << std::endl;
    std::cout << "Trying: " << next->endpoint() << std::endl;
    return next;
  }
};
```

It would be used with the `boost::asio::connect` function as follows:

```
tcp::resolver r(io_service);
tcp::resolver::query q("host", "service");
tcp::socket s(io_service);

// ...

r.async_resolve(q, resolve_handler);

// ...

void resolve_handler(
    const boost::system::error_code& ec,
    tcp::resolver::iterator i)
{
  if (!ec)
  {
    boost::asio::async_connect(s, i,
        my_connect_condition(),
        connect_handler);
  }
}

// ...

void connect_handler(
    const boost::system::error_code& ec,
    tcp::resolver::iterator i)
{
  if (ec)
  {
    // An error occurred.
  }
  else
  {
    std::cout << "Connected to: " << i->endpoint() << std::endl;
  }
}
```

async_connect (4 of 4 overloads)

Asynchronously establishes a socket connection by trying each endpoint in a sequence.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ConnectCondition,
    typename ComposedConnectHandler>
void async_connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    Iterator end,
    ConnectCondition connect_condition,
    ComposedConnectHandler handler);
```

This function attempts to connect a socket to one of a sequence of endpoints. It does this by repeated calls to the socket's `async_connect` member function, once for each endpoint in the sequence, until a connection is successfully established.

Parameters

s — The socket to be connected. If the socket is already open, it will be closed.

begin — An iterator pointing to the start of a sequence of endpoints.

end — An iterator pointing to the end of a sequence of endpoints.

connect_condition — A function object that is called prior to each connection attempt. The signature of the function object must be:

```
Iterator connect_condition(
    const boost::system::error_code& ec,
    Iterator next);
```

The `ec` parameter contains the result from the most recent connect operation. Before the first connection attempt, `ec` is always set to indicate success. The `next` parameter is an iterator pointing to the next endpoint to be tried. The function object should return the next iterator, but is permitted to return a different iterator so that endpoints may be skipped. The implementation guarantees that the function object will never be called with the end iterator.

handler — The handler to be called when the connect operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  // Result of operation. if the sequence is empty, set to
  // boost::asio::error::not_found. Otherwise, contains the
  // error from the last connection attempt.
  const boost::system::error_code& error,

  // On success, an iterator denoting the successfully
  // connected endpoint. Otherwise, the end iterator.
  Iterator iterator
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

The following connect condition function object can be used to output information about the individual connection attempts:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
struct my_connect_condition
{
  template <typename Iterator>
  Iterator operator()(
      const boost::system::error_code& ec,
      Iterator next)
  {
    if (ec) std::cout << "Error: " << ec.message() << std::endl;
    std::cout << "Trying: " << next->endpoint() << std::endl;
    return next;
  }
};
```

It would be used with the `boost::asio::connect` function as follows:

```
tcp::resolver r(io_service);
tcp::resolver::query q("host", "service");
tcp::socket s(io_service);

// ...

r.async_resolve(q, resolve_handler);

// ...

void resolve_handler(
    const boost::system::error_code& ec,
    tcp::resolver::iterator i)
{
  if (!ec)
  {
    tcp::resolver::iterator end;
    boost::asio::async_connect(s, i, end,
        my_connect_condition(),
        connect_handler);
  }
}

// ...

void connect_handler(
    const boost::system::error_code& ec,
    tcp::resolver::iterator i)
{
  if (ec)
  {
    // An error occurred.
  }
  else
  {
    std::cout << "Connected to: " << i->endpoint() << std::endl;
  }
}
```

async_read

Start an asynchronous operation to read a certain amount of data from a stream.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename AsyncReadStream,
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read(
    AsyncReadStream & s,
    const MutableBufferSequence & buffers,
    ReadHandler handler);
  » more...

template<
    typename AsyncReadStream,
    typename MutableBufferSequence,
    typename CompletionCondition,
    typename ReadHandler>
void async_read(
    AsyncReadStream & s,
    const MutableBufferSequence & buffers,
    CompletionCondition completion_condition,
    ReadHandler handler);
  » more...

template<
    typename AsyncReadStream,
    typename Allocator,
    typename ReadHandler>
void async_read(
    AsyncReadStream & s,
    basic_streambuf< Allocator > & b,
    ReadHandler handler);
  » more...

template<
    typename AsyncReadStream,
    typename Allocator,
    typename CompletionCondition,
    typename ReadHandler>
void async_read(
    AsyncReadStream & s,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    ReadHandler handler);
  » more...
```

Requirements

Header: `boost/asio/read.hpp`

Convenience header: `boost/asio.hpp`

async_read (1 of 4 overloads)

Start an asynchronous operation to read a certain amount of data from a stream.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename AsyncReadStream,
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read(
    AsyncReadStream & s,
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

This function is used to asynchronously read a certain number of bytes of data from a stream. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- The supplied buffers are full. That is, the bytes transferred is equal to the sum of the buffer sizes.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's async_read_some function, and is known as a **composed operation**. The program must ensure that the stream performs no other read operations (such as async_read, the stream's async_read_some function, or any other composed operations that perform reads) until this operation completes.

Parameters

s — The stream from which the data is to be read. The type must support the AsyncReadStream concept.

buffers — One or more buffers into which the data will be read. The sum of the buffer sizes indicates the maximum number of bytes to read from the stream. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.

  std::size_t bytes_transferred           // Number of bytes copied into the
                                          // buffers. If an error occurred,
                                          // this will be the  number of
                                          // bytes successfully transferred
                                          // prior to the error.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To read into a single data buffer use the `buffer` function as follows:

```
boost::asio::async_read(s, boost::asio::buffer(data, size), handler);
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

Remarks

This overload is equivalent to calling:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::async_read(
    s, buffers,
    boost::asio::transfer_all(),
    handler);
```

async_read (2 of 4 overloads)

Start an asynchronous operation to read a certain amount of data from a stream.

```
template<
    typename AsyncReadStream,
    typename MutableBufferSequence,
    typename CompletionCondition,
    typename ReadHandler>
void async_read(
    AsyncReadStream & s,
    const MutableBufferSequence & buffers,
    CompletionCondition completion_condition,
    ReadHandler handler);
```

This function is used to asynchronously read a certain number of bytes of data from a stream. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- The supplied buffers are full. That is, the bytes transferred is equal to the sum of the buffer sizes.
- The completion_condition function object returns 0.

Parameters

s — The stream from which the data is to be read. The type must support the AsyncReadStream concept.

buffers — One or more buffers into which the data will be read. The sum of the buffer sizes indicates the maximum number of bytes to read from the stream. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

completion_condition — The function object to be called to determine whether the read operation is complete. The signature of the function object must be:

```
std::size_t completion_condition(
  // Result of latest async_read_some operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the read operation is complete. A non-zero return value indicates the maximum number of bytes to be read on the next call to the stream's async_read_some function.

handler — The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void handler(
  const boost::system::error_code& error, // Result of operation.

  std::size_t bytes_transferred           // Number of bytes copied in↵
to the
                                          // buffers. If an error oc↵
curred,
                                         // this will be the  number of
                                          // bytes successfully trans↵
ferred
                                          // prior to the error.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To read into a single data buffer use the `buffer` function as follows:

```
boost::asio::async_read(s,
    boost::asio::buffer(data, size),
    boost::asio::transfer_at_least(32),
    handler);
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

async_read (3 of 4 overloads)

Start an asynchronous operation to read a certain amount of data from a stream.

```
template<
    typename AsyncReadStream,
    typename Allocator,
    typename ReadHandler>
void async_read(
    AsyncReadStream & s,
    basic_streambuf< Allocator > & b,
    ReadHandler handler);
```

This function is used to asynchronously read a certain number of bytes of data from a stream. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- The supplied buffer is full (that is, it has reached maximum size).
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's async_read_some function, and is known as a **composed operation**. The program must ensure that the stream performs no other read operations (such as async_read, the stream's async_read_some function, or any other composed operations that perform reads) until this operation completes.

Parameters

s — The stream from which the data is to be read. The type must support the AsyncReadStream concept.

b — A `basic_streambuf` object into which the data will be read. Ownership of the streambuf is retained by the caller, which must guarantee that it remains valid until the handler is called.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

handler The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.

  std::size_t bytes_transferred           // Number of bytes copied into the
                                          // buffers. If an error occurred,
                                          // this will be the  number of
                                          // bytes successfully transferred
                                          // prior to the error.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

This overload is equivalent to calling:

```
boost::asio::async_read(
    s, b,
    boost::asio::transfer_all(),
    handler);
```

async_read (4 of 4 overloads)

Start an asynchronous operation to read a certain amount of data from a stream.

```
template<
    typename AsyncReadStream,
    typename Allocator,
    typename CompletionCondition,
    typename ReadHandler>
void async_read(
    AsyncReadStream & s,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    ReadHandler handler);
```

This function is used to asynchronously read a certain number of bytes of data from a stream. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- The supplied buffer is full (that is, it has reached maximum size).
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the stream's async_read_some function, and is known as a **composed operation**. The program must ensure that the stream performs no other read operations (such as async_read, the stream's async_read_some function, or any other composed operations that perform reads) until this operation completes.

Parameters

s The stream from which the data is to be read. The type must support the AsyncReadStream concept.

b A `basic_streambuf` object into which the data will be read. Ownership of the streambuf is retained by the caller, which must guarantee that it remains valid until the handler is called.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

completion_condition

The function object to be called to determine whether the read operation is complete. The signature of the function object must be:

```
std::size_t completion_condition(
  // Result of latest async_read_some operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the read operation is complete. A non-zero return value indicates the maximum number of bytes to be read on the next call to the stream's async_read_some function.

handler

The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.

  std::size_t bytes_transferred           // Number of bytes copied in↵
to the
                                          // buffers. If an error oc↵
curred,
                                         // this will be the  number of
                                          // bytes successfully trans↵
ferred
                                          // prior to the error.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

async_read_at

Start an asynchronous operation to read a certain amount of data at the specified offset.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename AsyncRandomAccessReadDevice,
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_at(
    AsyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    ReadHandler handler);
  » more...

template<
    typename AsyncRandomAccessReadDevice,
    typename MutableBufferSequence,
    typename CompletionCondition,
    typename ReadHandler>
void async_read_at(
    AsyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    CompletionCondition completion_condition,
    ReadHandler handler);
  » more...

template<
    typename AsyncRandomAccessReadDevice,
    typename Allocator,
    typename ReadHandler>
void async_read_at(
    AsyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    ReadHandler handler);
  » more...

template<
    typename AsyncRandomAccessReadDevice,
    typename Allocator,
    typename CompletionCondition,
    typename ReadHandler>
void async_read_at(
    AsyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    ReadHandler handler);
  » more...
```

Requirements

Header: `boost/asio/read_at.hpp`

Convenience header: `boost/asio.hpp`

async_read_at (1 of 4 overloads)

Start an asynchronous operation to read a certain amount of data at the specified offset.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename AsyncRandomAccessReadDevice,
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_at(
    AsyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

This function is used to asynchronously read a certain number of bytes of data from a random access device at the specified offset. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- The supplied buffers are full. That is, the bytes transferred is equal to the sum of the buffer sizes.
- An error occurred.

This operation is implemented in terms of zero or more calls to the device's async_read_some_at function.

Parameters

d — The device from which the data is to be read. The type must support the AsyncRandomAccessReadDevice concept.

offset — The offset at which the data will be read.

buffers — One or more buffers into which the data will be read. The sum of the buffer sizes indicates the maximum number of bytes to read from the device. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  // Result of operation.
  const boost::system::error_code& error,

  // Number of bytes copied into the buffers. If an error
  // occurred, this will be the number of bytes successfully
  // transferred prior to the error.
  std::size_t bytes_transferred
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To read into a single data buffer use the `buffer` function as follows:

```
boost::asio::async_read_at(d, 42, boost::asio::buffer(data, size), handler);
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

Remarks

This overload is equivalent to calling:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::async_read_at(
    d, 42, buffers,
    boost::asio::transfer_all(),
    handler);
```

async_read_at (2 of 4 overloads)

Start an asynchronous operation to read a certain amount of data at the specified offset.

```
template<
    typename AsyncRandomAccessReadDevice,
    typename MutableBufferSequence,
    typename CompletionCondition,
    typename ReadHandler>
void async_read_at(
    AsyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    CompletionCondition completion_condition,
    ReadHandler handler);
```

This function is used to asynchronously read a certain number of bytes of data from a random access device at the specified offset. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- The supplied buffers are full. That is, the bytes transferred is equal to the sum of the buffer sizes.
- The completion_condition function object returns 0.

Parameters

d — The device from which the data is to be read. The type must support the AsyncRandomAccessReadDevice concept.

offset — The offset at which the data will be read.

buffers — One or more buffers into which the data will be read. The sum of the buffer sizes indicates the maximum number of bytes to read from the device. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

completion_condition — The function object to be called to determine whether the read operation is complete. The signature of the function object must be:

```
std::size_t completion_condition(
  // Result of latest async_read_some_at operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the read operation is complete. A non-zero return value indicates the maximum number of bytes to be read on the next call to the device's async_read_some_at function.

handler — The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void handler(
  // Result of operation.
  const boost::system::error_code& error,

  // Number of bytes copied into the buffers. If an error
  // occurred, this will be the number of bytes successfully
  // transferred prior to the error.
  std::size_t bytes_transferred
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To read into a single data buffer use the `buffer` function as follows:

```
boost::asio::async_read_at(d, 42,
    boost::asio::buffer(data, size),
    boost::asio::transfer_at_least(32),
    handler);
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

async_read_at (3 of 4 overloads)

Start an asynchronous operation to read a certain amount of data at the specified offset.

```
template<
    typename AsyncRandomAccessReadDevice,
    typename Allocator,
    typename ReadHandler>
void async_read_at(
    AsyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    ReadHandler handler);
```

This function is used to asynchronously read a certain number of bytes of data from a random access device at the specified offset. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- An error occurred.

This operation is implemented in terms of zero or more calls to the device's async_read_some_at function.

Parameters

d	The device from which the data is to be read. The type must support the AsyncRandomAccessReadDevice concept.
offset	The offset at which the data will be read.
b	A `basic_streambuf` object into which the data will be read. Ownership of the streambuf is retained by the caller, which must guarantee that it remains valid until the handler is called.
handler	The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void handler(
  // Result of operation.
  const boost::system::error_code& error,

  // Number of bytes copied into the buffers. If an error
  // occurred, this will be the number of bytes successfully
  // transferred prior to the error.
  std::size_t bytes_transferred
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

This overload is equivalent to calling:

```
boost::asio::async_read_at(
    d, 42, b,
    boost::asio::transfer_all(),
    handler);
```

async_read_at (4 of 4 overloads)

Start an asynchronous operation to read a certain amount of data at the specified offset.

```
template<
    typename AsyncRandomAccessReadDevice,
    typename Allocator,
    typename CompletionCondition,
    typename ReadHandler>
void async_read_at(
    AsyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    ReadHandler handler);
```

This function is used to asynchronously read a certain number of bytes of data from a random access device at the specified offset. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the device's async_read_some_at function.

Parameters

d	The device from which the data is to be read. The type must support the AsyncRandomAccessReadDevice concept.
offset	The offset at which the data will be read.
b	A `basic_streambuf` object into which the data will be read. Ownership of the streambuf is retained by the caller, which must guarantee that it remains valid until the handler is called.
completion_condition	The function object to be called to determine whether the read operation is complete. The signature of the function object must be:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t completion_condition(
  // Result of latest async_read_some_at operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the read operation is complete. A non-zero return value indicates the maximum number of bytes to be read on the next call to the device's async_read_some_at function.

handler

The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  // Result of operation.
  const boost::system::error_code& error,

  // Number of bytes copied into the buffers. If an error
  // occurred, this will be the number of bytes successfully
  // transferred prior to the error.
  std::size_t bytes_transferred
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

async_read_until

Start an asynchronous operation to read data into a streambuf until it contains a delimiter, matches a regular expression, or a function object indicates a match.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename AsyncReadStream,
    typename Allocator,
    typename ReadHandler>
void async_read_until(
    AsyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    char delim,
    ReadHandler handler);
  » more...

template<
    typename AsyncReadStream,
    typename Allocator,
    typename ReadHandler>
void async_read_until(
    AsyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    const std::string & delim,
    ReadHandler handler);
  » more...

template<
    typename AsyncReadStream,
    typename Allocator,
    typename ReadHandler>
void async_read_until(
    AsyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    const boost::regex & expr,
    ReadHandler handler);
  » more...

template<
    typename AsyncReadStream,
    typename Allocator,
    typename MatchCondition,
    typename ReadHandler>
void async_read_until(
    AsyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    MatchCondition match_condition,
    ReadHandler handler,
    typename boost::enable_if< is_match_condition< MatchCondition > >::type *  = 0);
  » more...
```

Requirements

Header: `boost/asio/read_until.hpp`

Convenience header: `boost/asio.hpp`

async_read_until (1 of 4 overloads)

Start an asynchronous operation to read data into a streambuf until it contains a specified delimiter.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename AsyncReadStream,
    typename Allocator,
    typename ReadHandler>
void async_read_until(
    AsyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    char delim,
    ReadHandler handler);
```

This function is used to asynchronously read data into the specified streambuf until the streambuf's get area contains the specified delimiter. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- The get area of the streambuf contains the specified delimiter.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's async_read_some function, and is known as a **composed operation**. If the streambuf's get area already contains the delimiter, this asynchronous operation completes immediately. The program must ensure that the stream performs no other read operations (such as async_read, async_read_until, the stream's async_read_some function, or any other composed operations that perform reads) until this operation completes.

Parameters

s — The stream from which the data is to be read. The type must support the AsyncReadStream concept.

b — A streambuf object into which the data will be read. Ownership of the streambuf is retained by the caller, which must guarantee that it remains valid until the handler is called.

delim — The delimiter character.

handler — The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  // Result of operation.
  const boost::system::error_code& error,

  // The number of bytes in the streambuf's get
  // area up to and including the delimiter.
  // 0 if an error occurred.
  std::size_t bytes_transferred
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

After a successful async_read_until operation, the streambuf may contain additional data beyond the delimiter. An application will typically leave that data in the streambuf for a subsequent async_read_until operation to examine.

Example

To asynchronously read data into a streambuf until a newline is encountered:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::streambuf b;
...
void handler(const boost::system::error_code& e, std::size_t size)
{
  if (!e)
  {
    std::istream is(&b);
    std::string line;
    std::getline(is, line);
    ...
  }
}
...
boost::asio::async_read_until(s, b, '\n', handler);
```

After the `async_read_until` operation completes successfully, the buffer `b` contains the delimiter:

```
{ 'a', 'b', ..., 'c', '\n', 'd', 'e', ... }
```

The call to `std::getline` then extracts the data up to and including the delimiter, so that the string `line` contains:

```
{ 'a', 'b', ..., 'c', '\n' }
```

The remaining data is left in the buffer `b` as follows:

```
{ 'd', 'e', ... }
```

This data may be the start of a new line, to be extracted by a subsequent `async_read_until` operation.

async_read_until (2 of 4 overloads)

Start an asynchronous operation to read data into a streambuf until it contains a specified delimiter.

```
template<
    typename AsyncReadStream,
    typename Allocator,
    typename ReadHandler>
void async_read_until(
    AsyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    const std::string & delim,
    ReadHandler handler);
```

This function is used to asynchronously read data into the specified streambuf until the streambuf's get area contains the specified delimiter. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- The get area of the streambuf contains the specified delimiter.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's async_read_some function, and is known as a **composed operation**. If the streambuf's get area already contains the delimiter, this asynchronous operation completes immediately. The program must ensure that the stream performs no other read operations (such as async_read, async_read_until, the stream's async_read_some function, or any other composed operations that perform reads) until this operation completes.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

s — The stream from which the data is to be read. The type must support the AsyncReadStream concept.

b — A streambuf object into which the data will be read. Ownership of the streambuf is retained by the caller, which must guarantee that it remains valid until the handler is called.

delim — The delimiter string.

handler — The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  // Result of operation.
  const boost::system::error_code& error,

  // The number of bytes in the streambuf's get
  // area up to and including the delimiter.
  // 0 if an error occurred.
  std::size_t bytes_transferred
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

After a successful async_read_until operation, the streambuf may contain additional data beyond the delimiter. An application will typically leave that data in the streambuf for a subsequent async_read_until operation to examine.

Example

To asynchronously read data into a streambuf until a newline is encountered:

```
boost::asio::streambuf b;
...
void handler(const boost::system::error_code& e, std::size_t size)
{
  if (!e)
  {
    std::istream is(&b);
    std::string line;
    std::getline(is, line);
    ...
  }
}
...
boost::asio::async_read_until(s, b, "\r\n", handler);
```

After the `async_read_until` operation completes successfully, the buffer `b` contains the delimiter:

```
{ 'a', 'b', ..., 'c', '\r', '\n', 'd', 'e', ... }
```

The call to `std::getline` then extracts the data up to and including the delimiter, so that the string `line` contains:

```
{ 'a', 'b', ..., 'c', '\r', '\n' }
```

The remaining data is left in the buffer `b` as follows:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
{ 'd', 'e', ... }
```

This data may be the start of a new line, to be extracted by a subsequent `async_read_until` operation.

async_read_until (3 of 4 overloads)

Start an asynchronous operation to read data into a streambuf until some part of its data matches a regular expression.

```
template<
    typename AsyncReadStream,
    typename Allocator,
    typename ReadHandler>
void async_read_until(
    AsyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    const boost::regex & expr,
    ReadHandler handler);
```

This function is used to asynchronously read data into the specified streambuf until the streambuf's get area contains some data that matches a regular expression. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- A substring of the streambuf's get area matches the regular expression.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's async_read_some function, and is known as a **composed operation**. If the streambuf's get area already contains data that matches the regular expression, this asynchronous operation completes immediately. The program must ensure that the stream performs no other read operations (such as async_read, async_read_until, the stream's async_read_some function, or any other composed operations that perform reads) until this operation completes.

Parameters

s — The stream from which the data is to be read. The type must support the AsyncReadStream concept.

b — A streambuf object into which the data will be read. Ownership of the streambuf is retained by the caller, which must guarantee that it remains valid until the handler is called.

expr — The regular expression.

handler — The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  // Result of operation.
  const boost::system::error_code& error,

  // The number of bytes in the streambuf's get
  // area up to and including the substring
  // that matches the regular. expression.
  // 0 if an error occurred.
  std::size_t bytes_transferred
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

After a successful async_read_until operation, the streambuf may contain additional data beyond that which matched the regular expression. An application will typically leave that data in the streambuf for a subsequent async_read_until operation to examine.

Example

To asynchronously read data into a streambuf until a CR-LF sequence is encountered:

```
boost::asio::streambuf b;
...
void handler(const boost::system::error_code& e, std::size_t size)
{
  if (!e)
  {
    std::istream is(&b);
    std::string line;
    std::getline(is, line);
    ...
  }
}
...
boost::asio::async_read_until(s, b, boost::regex("\r\n"), handler);
```

After the `async_read_until` operation completes successfully, the buffer `b` contains the data which matched the regular expression:

```
{ 'a', 'b', ..., 'c', '\r', '\n', 'd', 'e', ... }
```

The call to `std::getline` then extracts the data up to and including the match, so that the string `line` contains:

```
{ 'a', 'b', ..., 'c', '\r', '\n' }
```

The remaining data is left in the buffer `b` as follows:

```
{ 'd', 'e', ... }
```

This data may be the start of a new line, to be extracted by a subsequent `async_read_until` operation.

async_read_until (4 of 4 overloads)

Start an asynchronous operation to read data into a streambuf until a function object indicates a match.

```
template<
    typename AsyncReadStream,
    typename Allocator,
    typename MatchCondition,
    typename ReadHandler>
void async_read_until(
    AsyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    MatchCondition match_condition,
    ReadHandler handler,
    typename boost::enable_if< is_match_condition< MatchCondition > >::type *  = 0);
```

This function is used to asynchronously read data into the specified streambuf until a user-defined match condition function object, when applied to the data contained in the streambuf, indicates a successful match. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- The match condition function object returns a std::pair where the second element evaluates to true.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's async_read_some function, and is known as a **composed operation**. If the match condition function object already indicates a match, this asynchronous operation completes immediately. The program must ensure that the stream performs no other read operations (such as async_read, async_read_until, the stream's async_read_some function, or any other composed operations that perform reads) until this operation completes.

Parameters

s — The stream from which the data is to be read. The type must support the AsyncReadStream concept.

b — A streambuf object into which the data will be read.

match_condition — The function object to be called to determine whether a match exists. The signature of the function object must be:

```
pair<iterator, bool> match_condition(iterator begin, iterator end);
```

where `iterator` represents the type:

```
buffers_iterator<basic_streambuf<Allocator>::const_buffers_type>
```

The iterator parameters `begin` and `end` define the range of bytes to be scanned to determine whether there is a match. The `first` member of the return value is an iterator marking one-past-the-end of the bytes that have been consumed by the match function. This iterator is used to calculate the `begin` parameter for any subsequent invocation of the match condition. The `second` member of the return value is true if a match has been found, false otherwise.

handler — The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  // Result of operation.
  const boost::system::error_code& error,

  // The number of bytes in the streambuf's get
  // area that have been fully consumed by the
  // match function. O if an error occurred.
  std::size_t bytes_transferred
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

After a successful async_read_until operation, the streambuf may contain additional data beyond that which matched the function object. An application will typically leave that data in the streambuf for a subsequent async_read_until operation to examine.

The default implementation of the `is_match_condition` type trait evaluates to true for function pointers and function objects with a `result_type` typedef. It must be specialised for other user-defined function objects.

Examples

To asynchronously read data into a streambuf until whitespace is encountered:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef boost::asio::buffers_iterator<
    boost::asio::streambuf::const_buffers_type> iterator;

std::pair<iterator, bool>
match_whitespace(iterator begin, iterator end)
{
  iterator i = begin;
  while (i != end)
    if (std::isspace(*i++))
      return std::make_pair(i, true);
  return std::make_pair(i, false);
}
...
void handler(const boost::system::error_code& e, std::size_t size);
...
boost::asio::streambuf b;
boost::asio::async_read_until(s, b, match_whitespace, handler);
```

To asynchronously read data into a streambuf until a matching character is found:

```
class match_char
{
public:
  explicit match_char(char c) : c_(c) {}

  template <typename Iterator>
  std::pair<Iterator, bool> operator()(
      Iterator begin, Iterator end) const
  {
    Iterator i = begin;
    while (i != end)
      if (c_ == *i++)
        return std::make_pair(i, true);
    return std::make_pair(i, false);
  }

private:
  char c_;
};

namespace asio {
  template <> struct is_match_condition<match_char>
    : public boost::true_type {};
} // namespace asio
...
void handler(const boost::system::error_code& e, std::size_t size);
...
boost::asio::streambuf b;
boost::asio::async_read_until(s, b, match_char('a'), handler);
```

async_write

Start an asynchronous operation to write a certain amount of data to a stream.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename AsyncWriteStream,
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write(
    AsyncWriteStream & s,
    const ConstBufferSequence & buffers,
    WriteHandler handler);
  » more...

template<
    typename AsyncWriteStream,
    typename ConstBufferSequence,
    typename CompletionCondition,
    typename WriteHandler>
void async_write(
    AsyncWriteStream & s,
    const ConstBufferSequence & buffers,
    CompletionCondition completion_condition,
    WriteHandler handler);
  » more...

template<
    typename AsyncWriteStream,
    typename Allocator,
    typename WriteHandler>
void async_write(
    AsyncWriteStream & s,
    basic_streambuf< Allocator > & b,
    WriteHandler handler);
  » more...

template<
    typename AsyncWriteStream,
    typename Allocator,
    typename CompletionCondition,
    typename WriteHandler>
void async_write(
    AsyncWriteStream & s,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    WriteHandler handler);
  » more...
```

Requirements

Header: `boost/asio/write.hpp`

Convenience header: `boost/asio.hpp`

async_write (1 of 4 overloads)

Start an asynchronous operation to write all of the supplied data to a stream.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename AsyncWriteStream,
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write(
    AsyncWriteStream & s,
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

This function is used to asynchronously write a certain number of bytes of data to a stream. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- All of the data in the supplied buffers has been written. That is, the bytes transferred is equal to the sum of the buffer sizes.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's async_write_some function, and is known as a **composed operation**. The program must ensure that the stream performs no other write operations (such as async_write, the stream's async_write_some function, or any other composed operations that perform writes) until this operation completes.

Parameters

s — The stream to which the data is to be written. The type must support the AsyncWriteStream concept.

buffers — One or more buffers containing the data to be written. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the write operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.

  std::size_t bytes_transferred           // Number of bytes written from the
                                          // buffers. If an error occurred,
                                          // this will be less than the sum
                                          // of the buffer sizes.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To write a single data buffer use the `buffer` function as follows:

```
boost::asio::async_write(s, boost::asio::buffer(data, size), handler);
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

async_write (2 of 4 overloads)

Start an asynchronous operation to write a certain amount of data to a stream.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename AsyncWriteStream,
    typename ConstBufferSequence,
    typename CompletionCondition,
    typename WriteHandler>
void async_write(
    AsyncWriteStream & s,
    const ConstBufferSequence & buffers,
    CompletionCondition completion_condition,
    WriteHandler handler);
```

This function is used to asynchronously write a certain number of bytes of data to a stream. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- All of the data in the supplied buffers has been written. That is, the bytes transferred is equal to the sum of the buffer sizes.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the stream's async_write_some function, and is known as a **composed operation**. The program must ensure that the stream performs no other write operations (such as async_write, the stream's async_write_some function, or any other composed operations that perform writes) until this operation completes.

Parameters

s — The stream to which the data is to be written. The type must support the AsyncWriteStream concept.

buffers — One or more buffers containing the data to be written. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

completion_condition — The function object to be called to determine whether the write operation is complete. The signature of the function object must be:

```
std::size_t completion_condition(
  // Result of latest async_write_some operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the write operation is complete. A non-zero return value indicates the maximum number of bytes to be written on the next call to the stream's async_write_some function.

handler — The handler to be called when the write operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.

  std::size_t bytes_transferred           // Number of bytes written ↵
from the
                                          // buffers. If an error oc↵
curred,
                                        // this will be less than the ↵
sum
                                          // of the buffer sizes.
);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To write a single data buffer use the `buffer` function as follows:

```
boost::asio::async_write(s,
    boost::asio::buffer(data, size),
    boost::asio::transfer_at_least(32),
    handler);
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

async_write (3 of 4 overloads)

Start an asynchronous operation to write all of the supplied data to a stream.

```
template<
    typename AsyncWriteStream,
    typename Allocator,
    typename WriteHandler>
void async_write(
    AsyncWriteStream & s,
    basic_streambuf< Allocator > & b,
    WriteHandler handler);
```

This function is used to asynchronously write a certain number of bytes of data to a stream. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- All of the data in the supplied `basic_streambuf` has been written.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's async_write_some function, and is known as a **composed operation**. The program must ensure that the stream performs no other write operations (such as async_write, the stream's async_write_some function, or any other composed operations that perform writes) until this operation completes.

Parameters

s — The stream to which the data is to be written. The type must support the AsyncWriteStream concept.

b — A `basic_streambuf` object from which data will be written. Ownership of the streambuf is retained by the caller, which must guarantee that it remains valid until the handler is called.

handler — The handler to be called when the write operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.

  std::size_t bytes_transferred           // Number of bytes written from the
                                          // buffers. If an error occurred,
                                          // this will be less than the sum
                                          // of the buffer sizes.
);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

async_write (4 of 4 overloads)

Start an asynchronous operation to write a certain amount of data to a stream.

```
template<
    typename AsyncWriteStream,
    typename Allocator,
    typename CompletionCondition,
    typename WriteHandler>
void async_write(
    AsyncWriteStream & s,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    WriteHandler handler);
```

This function is used to asynchronously write a certain number of bytes of data to a stream. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- All of the data in the supplied `basic_streambuf` has been written.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the stream's async_write_some function, and is known as a **composed operation**. The program must ensure that the stream performs no other write operations (such as async_write, the stream's async_write_some function, or any other composed operations that perform writes) until this operation completes.

Parameters

s — The stream to which the data is to be written. The type must support the AsyncWriteStream concept.

b — A `basic_streambuf` object from which data will be written. Ownership of the streambuf is retained by the caller, which must guarantee that it remains valid until the handler is called.

completion_condition — The function object to be called to determine whether the write operation is complete. The signature of the function object must be:

```
std::size_t completion_condition(
  // Result of latest async_write_some operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the write operation is complete. A non-zero return value indicates the maximum number of bytes to be written on the next call to the stream's async_write_some function.

handler — The handler to be called when the write operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void handler(
  const boost::system::error_code& error, // Result of operation.

  std::size_t bytes_transferred           // Number of bytes written ↵
from the
                                          // buffers. If an error oc↵
curred,
                                        // this will be less than the ↵
sum
                                          // of the buffer sizes.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

async_write_at

Start an asynchronous operation to write a certain amount of data at the specified offset.

```
template<
    typename AsyncRandomAccessWriteDevice,
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_at(
    AsyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    WriteHandler handler);
  » more...

template<
    typename AsyncRandomAccessWriteDevice,
    typename ConstBufferSequence,
    typename CompletionCondition,
    typename WriteHandler>
void async_write_at(
    AsyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    CompletionCondition completion_condition,
    WriteHandler handler);
  » more...

template<
    typename AsyncRandomAccessWriteDevice,
    typename Allocator,
    typename WriteHandler>
void async_write_at(
    AsyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    WriteHandler handler);
  » more...

template<
    typename AsyncRandomAccessWriteDevice,
    typename Allocator,
    typename CompletionCondition,
    typename WriteHandler>
void async_write_at(
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    AsyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    WriteHandler handler);
  » more...
```

Requirements

Header: `boost/asio/write_at.hpp`

Convenience header: `boost/asio.hpp`

async_write_at (1 of 4 overloads)

Start an asynchronous operation to write all of the supplied data at the specified offset.

```
template<
    typename AsyncRandomAccessWriteDevice,
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_at(
    AsyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

This function is used to asynchronously write a certain number of bytes of data to a random access device at a specified offset. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- All of the data in the supplied buffers has been written. That is, the bytes transferred is equal to the sum of the buffer sizes.
- An error occurred.

This operation is implemented in terms of zero or more calls to the device's async_write_some_at function.

Parameters

d — The device to which the data is to be written. The type must support the AsyncRandomAccessWriteDevice concept.

offset — The offset at which the data will be written.

buffers — One or more buffers containing the data to be written. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the write operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  // Result of operation.
  const boost::system::error_code& error,

  // Number of bytes written from the buffers. If an error
  // occurred, this will be less than the sum of the buffer sizes.
  std::size_t bytes_transferred
);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To write a single data buffer use the `buffer` function as follows:

```
boost::asio::async_write_at(d, 42, boost::asio::buffer(data, size), handler);
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

async_write_at (2 of 4 overloads)

Start an asynchronous operation to write a certain amount of data at the specified offset.

```
template<
    typename AsyncRandomAccessWriteDevice,
    typename ConstBufferSequence,
    typename CompletionCondition,
    typename WriteHandler>
void async_write_at(
    AsyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    CompletionCondition completion_condition,
    WriteHandler handler);
```

This function is used to asynchronously write a certain number of bytes of data to a random access device at a specified offset. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- All of the data in the supplied buffers has been written. That is, the bytes transferred is equal to the sum of the buffer sizes.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the device's async_write_some_at function.

Parameters

d	The device to which the data is to be written. The type must support the AsyncRandomAccessWriteDevice concept.
offset	The offset at which the data will be written.
buffers	One or more buffers containing the data to be written. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.
completion_condition	The function object to be called to determine whether the write operation is complete. The signature of the function object must be:

```
std::size_t completion_condition(
  // Result of latest async_write_some_at operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

A return value of 0 indicates that the write operation is complete. A non-zero return value indicates the maximum number of bytes to be written on the next call to the device's async_write_some_at function.

handler

The handler to be called when the write operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  // Result of operation.
  const boost::system::error_code& error,

  // Number of bytes written from the buffers. If an error
  // occurred, this will be less than the sum of the buffer sizes.
  std::size_t bytes_transferred
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To write a single data buffer use the `buffer` function as follows:

```
boost::asio::async_write_at(d, 42,
    boost::asio::buffer(data, size),
    boost::asio::transfer_at_least(32),
    handler);
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

async_write_at (3 of 4 overloads)

Start an asynchronous operation to write all of the supplied data at the specified offset.

```
template<
    typename AsyncRandomAccessWriteDevice,
    typename Allocator,
    typename WriteHandler>
void async_write_at(
    AsyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    WriteHandler handler);
```

This function is used to asynchronously write a certain number of bytes of data to a random access device at a specified offset. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- All of the data in the supplied `basic_streambuf` has been written.
- An error occurred.

This operation is implemented in terms of zero or more calls to the device's async_write_some_at function.

Parameters

d

The device to which the data is to be written. The type must support the AsyncRandomAccessWriteDevice concept.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

offset — The offset at which the data will be written.

b — A `basic_streambuf` object from which data will be written. Ownership of the streambuf is retained by the caller, which must guarantee that it remains valid until the handler is called.

handler — The handler to be called when the write operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  // Result of operation.
  const boost::system::error_code& error,

  // Number of bytes written from the buffers. If an error
  // occurred, this will be less than the sum of the buffer sizes.
  std::size_t bytes_transferred
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

async_write_at (4 of 4 overloads)

Start an asynchronous operation to write a certain amount of data at the specified offset.

```
template<
    typename AsyncRandomAccessWriteDevice,
    typename Allocator,
    typename CompletionCondition,
    typename WriteHandler>
void async_write_at(
    AsyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    WriteHandler handler);
```

This function is used to asynchronously write a certain number of bytes of data to a random access device at a specified offset. The function call always returns immediately. The asynchronous operation will continue until one of the following conditions is true:

- All of the data in the supplied `basic_streambuf` has been written.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the device's async_write_some_at function.

Parameters

d — The device to which the data is to be written. The type must support the AsyncRandomAccessWriteDevice concept.

offset — The offset at which the data will be written.

b — A `basic_streambuf` object from which data will be written. Ownership of the streambuf is retained by the caller, which must guarantee that it remains valid until the handler is called.

completion_condition — The function object to be called to determine whether the write operation is complete. The signature of the function object must be:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t completion_condition(
  // Result of latest async_write_some_at operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the write operation is complete. A non-zero return value indicates the maximum number of bytes to be written on the next call to the device's async_write_some_at function.

handler

The handler to be called when the write operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  // Result of operation.
  const boost::system::error_code& error,

  // Number of bytes written from the buffers. If an error
  // occurred, this will be less than the sum of the buffer sizes.
  std::size_t bytes_transferred
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

basic_datagram_socket

Provides datagram-oriented socket functionality.

```
template<
    typename Protocol,
    typename DatagramSocketService = datagram_socket_service<Protocol>>
class basic_datagram_socket :
  public basic_socket< Protocol, DatagramSocketService >
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
async_receive	Start an asynchronous receive on a connected socket.
async_receive_from	Start an asynchronous receive.
async_send	Start an asynchronous send on a connected socket.
async_send_to	Start an asynchronous send.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_datagram_socket	Construct a basic_datagram_socket without opening it. Construct and open a basic_datagram_socket. Construct a basic_datagram_socket, opening it and binding it to the given local endpoint. Construct a basic_datagram_socket on an existing native socket. Move-construct a basic_datagram_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_datagram_socket from another.
receive	Receive some data on a connected socket.
receive_from	Receive a datagram with the endpoint of the sender.
remote_endpoint	Get the remote endpoint of the socket.
send	Send some data on a connected socket.
send_to	Send a datagram to the specified endpoint.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_datagram_socket` class template provides asynchronous and blocking datagram-oriented socket functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::assign

Assign an existing native socket to the socket.

```
void assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket);
  » more...

boost::system::error_code assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::assign (1 of 2 overloads)

Inherited from basic_socket.

Assign an existing native socket to the socket.

```
void assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket);
```

basic_datagram_socket::assign (2 of 2 overloads)

Inherited from basic_socket.

Assign an existing native socket to the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
```

basic_datagram_socket::async_connect

Inherited from basic_socket.

Start an asynchronous connect.

```
template<
    typename ConnectHandler>
void async_connect(
    const endpoint_type & peer_endpoint,
    ConnectHandler handler);
```

This function is used to asynchronously connect a socket to the specified remote endpoint. The function call always returns immediately.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint — The remote endpoint to which the socket will be connected. Copies will be made of the endpoint object as required.

handler — The handler to be called when the connection operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error // Result of operation
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

```
void connect_handler(const boost::system::error_code& error)
{
  if (!error)
  {
    // Connect succeeded.
  }
}

...

boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.async_connect(endpoint, connect_handler);
```

basic_datagram_socket::async_receive

Start an asynchronous receive on a connected socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    ReadHandler handler);
  » more...

template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    ReadHandler handler);
  » more...
```

basic_datagram_socket::async_receive (1 of 2 overloads)

Start an asynchronous receive on a connected socket.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

This function is used to asynchronously receive data from the datagram socket. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be received. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the receive operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes received.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The async_receive operation can only be used with a connected socket. Use the async_receive_from function to receive data on an unconnected datagram socket.

Example

To receive into a single data buffer use the `buffer` function as follows:

```
socket.async_receive(boost::asio::buffer(data, size), handler);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

See the buffer documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_datagram_socket::async_receive (2 of 2 overloads)

Start an asynchronous receive on a connected socket.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    ReadHandler handler);
```

This function is used to asynchronously receive data from the datagram socket. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be received. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

flags — Flags specifying how the receive call is to be made.

handler — The handler to be called when the receive operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes received.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The async_receive operation can only be used with a connected socket. Use the async_receive_from function to receive data on an unconnected datagram socket.

basic_datagram_socket::async_receive_from

Start an asynchronous receive.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    ReadHandler handler);
  » more...

template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags,
    ReadHandler handler);
  » more...
```

basic_datagram_socket::async_receive_from (1 of 2 overloads)

Start an asynchronous receive.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    ReadHandler handler);
```

This function is used to asynchronously receive a datagram. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be received. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

sender_endpoint — An endpoint object that receives the endpoint of the remote sender of the datagram. Ownership of the sender_endpoint object is retained by the caller, which must guarantee that it is valid until the handler is called.

handler — The handler to be called when the receive operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes received.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To receive into a single data buffer use the `buffer` function as follows:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
socket.async_receive_from(
    boost::asio::buffer(data, size), 0, sender_endpoint, handler);
```

See the buffer documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_datagram_socket::async_receive_from (2 of 2 overloads)

Start an asynchronous receive.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags,
    ReadHandler handler);
```

This function is used to asynchronously receive a datagram. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be received. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

sender_endpoint — An endpoint object that receives the endpoint of the remote sender of the datagram. Ownership of the sender_endpoint object is retained by the caller, which must guarantee that it is valid until the handler is called.

flags — Flags specifying how the receive call is to be made.

handler — The handler to be called when the receive operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes received.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

basic_datagram_socket::async_send

Start an asynchronous send on a connected socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    const ConstBufferSequence & buffers,
    WriteHandler handler);
  » more...

template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    WriteHandler handler);
  » more...
```

basic_datagram_socket::async_send (1 of 2 overloads)

Start an asynchronous send on a connected socket.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

This function is used to send data on the datagram socket. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers — One or more data buffers to be sent on the socket. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the send operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes sent.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The async_send operation can only be used with a connected socket. Use the async_send_to function to send data on an unconnected datagram socket.

Example

To send a single data buffer use the `buffer` function as follows:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
socket.async_send(boost::asio::buffer(data, size), handler);
```

See the buffer documentation for information on sending multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_datagram_socket::async_send (2 of 2 overloads)

Start an asynchronous send on a connected socket.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    WriteHandler handler);
```

This function is used to send data on the datagram socket. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers — One or more data buffers to be sent on the socket. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

flags — Flags specifying how the send call is to be made.

handler — The handler to be called when the send operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes sent.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The async_send operation can only be used with a connected socket. Use the async_send_to function to send data on an unconnected datagram socket.

basic_datagram_socket::async_send_to

Start an asynchronous send.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    WriteHandler handler);
  » more...

template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags,
    WriteHandler handler);
  » more...
```

basic_datagram_socket::async_send_to (1 of 2 overloads)

Start an asynchronous send.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    WriteHandler handler);
```

This function is used to asynchronously send a datagram to the specified remote endpoint. The function call always returns immediately.

Parameters

buffers — One or more data buffers to be sent to the remote endpoint. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

destination — The remote endpoint to which the data will be sent. Copies will be made of the endpoint as required.

handler — The handler to be called when the send operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes sent.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To send a single data buffer use the `buffer` function as follows:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::udp::endpoint destination(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.async_send_to(
    boost::asio::buffer(data, size), destination, handler);
```

See the buffer documentation for information on sending multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_datagram_socket::async_send_to (2 of 2 overloads)

Start an asynchronous send.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags,
    WriteHandler handler);
```

This function is used to asynchronously send a datagram to the specified remote endpoint. The function call always returns immediately.

Parameters

buffers — One or more data buffers to be sent to the remote endpoint. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

flags — Flags specifying how the send call is to be made.

destination — The remote endpoint to which the data will be sent. Copies will be made of the endpoint as required.

handler — The handler to be called when the send operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes sent.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

basic_datagram_socket::at_mark

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark() const;
  » more...

bool at_mark(
    boost::system::error_code & ec) const;
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_datagram_socket::at_mark (1 of 2 overloads)

Inherited from basic_socket.

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark() const;
```

This function is used to check whether the socket input is currently positioned at the out-of-band data mark.

Return Value

A bool indicating whether the socket is at the out-of-band data mark.

Exceptions

boost::system::system_error Thrown on failure.

basic_datagram_socket::at_mark (2 of 2 overloads)

Inherited from basic_socket.

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark(
    boost::system::error_code & ec) const;
```

This function is used to check whether the socket input is currently positioned at the out-of-band data mark.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

A bool indicating whether the socket is at the out-of-band data mark.

basic_datagram_socket::available

Determine the number of bytes available for reading.

```
std::size_t available() const;
  » more...

std::size_t available(
    boost::system::error_code & ec) const;
  » more...
```

basic_datagram_socket::available (1 of 2 overloads)

Inherited from basic_socket.

Determine the number of bytes available for reading.

```
std::size_t available() const;
```

This function is used to determine the number of bytes that may be read without blocking.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Return Value

The number of bytes that may be read without blocking, or 0 if an error occurs.

Exceptions

boost::system::system_error Thrown on failure.

basic_datagram_socket::available (2 of 2 overloads)

Inherited from basic_socket.

Determine the number of bytes available for reading.

```
std::size_t available(
    boost::system::error_code & ec) const;
```

This function is used to determine the number of bytes that may be read without blocking.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes that may be read without blocking, or 0 if an error occurs.

basic_datagram_socket::basic_datagram_socket

Construct a `basic_datagram_socket` without opening it.

```
explicit basic_datagram_socket(
    boost::asio::io_service & io_service);
  » more...
```

Construct and open a `basic_datagram_socket`.

```
basic_datagram_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol);
  » more...
```

Construct a `basic_datagram_socket`, opening it and binding it to the given local endpoint.

```
basic_datagram_socket(
    boost::asio::io_service & io_service,
    const endpoint_type & endpoint);
  » more...
```

Construct a `basic_datagram_socket` on an existing native socket.

```
basic_datagram_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol,
    const native_handle_type & native_socket);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Move-construct a basic_datagram_socket from another.

```
basic_datagram_socket(
    basic_datagram_socket && other);
  » more...
```

basic_datagram_socket::basic_datagram_socket (1 of 5 overloads)

Construct a basic_datagram_socket without opening it.

```
basic_datagram_socket(
    boost::asio::io_service & io_service);
```

This constructor creates a datagram socket without opening it. The open() function must be called before data can be sent or received on the socket.

Parameters

io_service — The io_service object that the datagram socket will use to dispatch handlers for any asynchronous operations performed on the socket.

basic_datagram_socket::basic_datagram_socket (2 of 5 overloads)

Construct and open a basic_datagram_socket.

```
basic_datagram_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol);
```

This constructor creates and opens a datagram socket.

Parameters

io_service — The io_service object that the datagram socket will use to dispatch handlers for any asynchronous operations performed on the socket.

protocol — An object specifying protocol parameters to be used.

Exceptions

boost::system::system_error — Thrown on failure.

basic_datagram_socket::basic_datagram_socket (3 of 5 overloads)

Construct a basic_datagram_socket, opening it and binding it to the given local endpoint.

```
basic_datagram_socket(
    boost::asio::io_service & io_service,
    const endpoint_type & endpoint);
```

This constructor creates a datagram socket and automatically opens it bound to the specified endpoint on the local machine. The protocol used is the protocol associated with the given endpoint.

Parameters

io_service — The io_service object that the datagram socket will use to dispatch handlers for any asynchronous operations performed on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

endpoint An endpoint on the local machine to which the datagram socket will be bound.

Exceptions

boost::system::system_error Thrown on failure.

basic_datagram_socket::basic_datagram_socket (4 of 5 overloads)

Construct a `basic_datagram_socket` on an existing native socket.

```
basic_datagram_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol,
    const native_handle_type & native_socket);
```

This constructor creates a datagram socket object to hold an existing native socket.

Parameters

io_service The `io_service` object that the datagram socket will use to dispatch handlers for any asynchronous operations performed on the socket.

protocol An object specifying protocol parameters to be used.

native_socket The new underlying socket implementation.

Exceptions

boost::system::system_error Thrown on failure.

basic_datagram_socket::basic_datagram_socket (5 of 5 overloads)

Move-construct a `basic_datagram_socket` from another.

```
basic_datagram_socket(
    basic_datagram_socket && other);
```

This constructor moves a datagram socket from one object to another.

Parameters

other The other `basic_datagram_socket` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_datagram_socket(io_service&)` `constructor`.

basic_datagram_socket::bind

Bind the socket to the given local endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void bind(
    const endpoint_type & endpoint);
  » more...

boost::system::error_code bind(
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::bind (1 of 2 overloads)

Inherited from basic_socket.

Bind the socket to the given local endpoint.

```
void bind(
    const endpoint_type & endpoint);
```

This function binds the socket to the specified endpoint on the local machine.

Parameters

endpoint An endpoint on the local machine to which the socket will be bound.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
socket.bind(boost::asio::ip::tcp::endpoint(
      boost::asio::ip::tcp::v4(), 12345));
```

basic_datagram_socket::bind (2 of 2 overloads)

Inherited from basic_socket.

Bind the socket to the given local endpoint.

```
boost::system::error_code bind(
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
```

This function binds the socket to the specified endpoint on the local machine.

Parameters

endpoint An endpoint on the local machine to which the socket will be bound.

ec Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
boost::system::error_code ec;
socket.bind(boost::asio::ip::tcp::endpoint(
      boost::asio::ip::tcp::v4(), 12345), ec);
if (ec)
{
  // An error occurred.
}
```

basic_datagram_socket::broadcast

Inherited from socket_base.

Socket option to permit sending of broadcast messages.

```
typedef implementation_defined broadcast;
```

Implements the SOL_SOCKET/SO_BROADCAST socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::bytes_readable

Inherited from socket_base.

IO control command to get the amount of data that can be read without blocking.

```
typedef implementation_defined bytes_readable;
```

Implements the FIONREAD IO control command.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::bytes_readable command(true);
socket.io_control(command);
std::size_t bytes_readable = command.get();
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::cancel

Cancel all asynchronous operations associated with the socket.

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::cancel (1 of 2 overloads)

Inherited from basic_socket.

Cancel all asynchronous operations associated with the socket.

```
void cancel();
```

This function causes all outstanding asynchronous connect, send and receive operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

Calls to `cancel()` will always fail with `boost::asio::error::operation_not_supported` when run on Windows XP, Windows Server 2003, and earlier versions of Windows, unless BOOST_ASIO_ENABLE_CANCELIO is defined. However, the CancelIo function has two issues that should be considered before enabling its use:

- It will only cancel asynchronous operations that were initiated in the current thread.
- It can appear to complete without error, but the request to cancel the unfinished operations may be silently ignored by the operating system. Whether it works or not seems to depend on the drivers that are installed.

For portable cancellation, consider using one of the following alternatives:

- Disable asio's I/O completion port backend by defining BOOST_ASIO_DISABLE_IOCP.
- Use the `close()` function to simultaneously cancel the outstanding operations and close the socket.

When running on Windows Vista, Windows Server 2008, and later, the CancelIoEx function is always used. This function does not have the problems described above.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_datagram_socket::cancel (2 of 2 overloads)

Inherited from basic_socket.

Cancel all asynchronous operations associated with the socket.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function causes all outstanding asynchronous connect, send and receive operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Parameters

ec Set to indicate what error occurred, if any.

Remarks

Calls to `cancel()` will always fail with `boost::asio::error::operation_not_supported` when run on Windows XP, Windows Server 2003, and earlier versions of Windows, unless BOOST_ASIO_ENABLE_CANCELIO is defined. However, the CancelIo function has two issues that should be considered before enabling its use:

- It will only cancel asynchronous operations that were initiated in the current thread.
- It can appear to complete without error, but the request to cancel the unfinished operations may be silently ignored by the operating system. Whether it works or not seems to depend on the drivers that are installed.

For portable cancellation, consider using one of the following alternatives:

- Disable asio's I/O completion port backend by defining BOOST_ASIO_DISABLE_IOCP.
- Use the `close()` function to simultaneously cancel the outstanding operations and close the socket.

When running on Windows Vista, Windows Server 2008, and later, the CancelIoEx function is always used. This function does not have the problems described above.

basic_datagram_socket::close

Close the socket.

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::close (1 of 2 overloads)

Inherited from basic_socket.

Close the socket.

```
void close();
```

This function is used to close the socket. Any asynchronous send, receive or connect operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure. Note that, even if the function indicates an error, the underlying descriptor is closed.

Remarks

For portable behaviour with respect to graceful closure of a connected socket, call `shutdown()` before closing the socket.

basic_datagram_socket::close (2 of 2 overloads)

Inherited from basic_socket.

Close the socket.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```

This function is used to close the socket. Any asynchronous send, receive or connect operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Parameters

ec	Set to indicate what error occurred, if any. Note that, even if the function indicates an error, the underlying descriptor is closed.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
socket.close(ec);
if (ec)
{
  // An error occurred.
}
```

Remarks

For portable behaviour with respect to graceful closure of a connected socket, call `shutdown()` before closing the socket.

basic_datagram_socket::connect

Connect the socket to the specified endpoint.

```
void connect(
    const endpoint_type & peer_endpoint);
  » more...

boost::system::error_code connect(
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::connect (1 of 2 overloads)

Inherited from basic_socket.

Connect the socket to the specified endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void connect(
    const endpoint_type & peer_endpoint);
```

This function is used to connect a socket to the specified remote endpoint. The function call will block until the connection is successfully made or an error occurs.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint	The remote endpoint to which the socket will be connected.

Exceptions

boost::system::system_error	Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.connect(endpoint);
```

basic_datagram_socket::connect (2 of 2 overloads)

Inherited from basic_socket.

Connect the socket to the specified endpoint.

```
boost::system::error_code connect(
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
```

This function is used to connect a socket to the specified remote endpoint. The function call will block until the connection is successfully made or an error occurs.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint	The remote endpoint to which the socket will be connected.
ec	Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
boost::system::error_code ec;
socket.connect(endpoint, ec);
if (ec)
{
  // An error occurred.
}
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_datagram_socket::debug

Inherited from socket_base.

Socket option to enable socket-level debugging.

```
typedef implementation_defined debug;
```

Implements the SOL_SOCKET/SO_DEBUG socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::do_not_route

Inherited from socket_base.

Socket option to prevent routing, use local interfaces only.

```
typedef implementation_defined do_not_route;
```

Implements the SOL_SOCKET/SO_DONTROUTE socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option(true);
socket.set_option(option);
```

Getting the current option value:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::enable_connection_aborted

Inherited from socket_base.

Socket option to report aborted connections on accept.

```
typedef implementation_defined enable_connection_aborted;
```

Implements a custom socket option that determines whether or not an accept operation is permitted to fail with `boost::asio::error::connection_aborted`. By default the option is false.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option(true);
acceptor.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::endpoint_type

The endpoint type.

```
typedef Protocol::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_datagram_socket::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

basic_datagram_socket::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

basic_datagram_socket::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

basic_datagram_socket::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

basic_datagram_socket::get_option

Get an option from the socket.

```
void get_option(
    GettableSocketOption & option) const;
  » more...

boost::system::error_code get_option(
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
  » more...
```

basic_datagram_socket::get_option (1 of 2 overloads)

Inherited from basic_socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Get an option from the socket.

```
template<
    typename GettableSocketOption>
void get_option(
    GettableSocketOption & option) const;
```

This function is used to get the current value of an option on the socket.

Parameters

option The option value to be obtained from the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Getting the value of the SOL_SOCKET/SO_KEEPALIVE option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::keep_alive option;
socket.get_option(option);
bool is_set = option.get();
```

basic_datagram_socket::get_option (2 of 2 overloads)

Inherited from basic_socket.

Get an option from the socket.

```
template<
    typename GettableSocketOption>
boost::system::error_code get_option(
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
```

This function is used to get the current value of an option on the socket.

Parameters

option The option value to be obtained from the socket.

ec Set to indicate what error occurred, if any.

Example

Getting the value of the SOL_SOCKET/SO_KEEPALIVE option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::keep_alive option;
boost::system::error_code ec;
socket.get_option(option, ec);
if (ec)
{
  // An error occurred.
}
bool is_set = option.get();
```

basic_datagram_socket::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

basic_datagram_socket::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

basic_datagram_socket::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

basic_datagram_socket::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

basic_datagram_socket::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::io_control

Perform an IO control command on the socket.

```
void io_control(
    IoControlCommand & command);
  » more...

boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::io_control (1 of 2 overloads)

Inherited from basic_socket.

Perform an IO control command on the socket.

```
template<
    typename IoControlCommand>
void io_control(
    IoControlCommand & command);
```

This function is used to execute an IO control command on the socket.

Parameters

command The IO control command to be performed on the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Getting the number of bytes ready to read:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::bytes_readable command;
socket.io_control(command);
std::size_t bytes_readable = command.get();
```

basic_datagram_socket::io_control (2 of 2 overloads)

Inherited from basic_socket.

Perform an IO control command on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
```

This function is used to execute an IO control command on the socket.

Parameters

command The IO control command to be performed on the socket.

ec Set to indicate what error occurred, if any.

Example

Getting the number of bytes ready to read:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::bytes_readable command;
boost::system::error_code ec;
socket.io_control(command, ec);
if (ec)
{
  // An error occurred.
}
std::size_t bytes_readable = command.get();
```

basic_datagram_socket::is_open

Inherited from basic_socket.

Determine whether the socket is open.

```
bool is_open() const;
```

basic_datagram_socket::keep_alive

Inherited from socket_base.

Socket option to send keep-alives.

```
typedef implementation_defined keep_alive;
```

Implements the SOL_SOCKET/SO_KEEPALIVE socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option(true);
socket.set_option(option);
```

Getting the current option value:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::linger

Inherited from socket_base.

Socket option to specify whether the socket lingers on close if unsent data is present.

```
typedef implementation_defined linger;
```

Implements the SOL_SOCKET/SO_LINGER socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option(true, 30);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option;
socket.get_option(option);
bool is_set = option.enabled();
unsigned short timeout = option.timeout();
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::local_endpoint

Get the local endpoint of the socket.

```
endpoint_type local_endpoint() const;
  » more...

endpoint_type local_endpoint(
    boost::system::error_code & ec) const;
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_datagram_socket::local_endpoint (1 of 2 overloads)

Inherited from basic_socket.

Get the local endpoint of the socket.

```
endpoint_type local_endpoint() const;
```

This function is used to obtain the locally bound endpoint of the socket.

Return Value

An object that represents the local endpoint of the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::endpoint endpoint = socket.local_endpoint();
```

basic_datagram_socket::local_endpoint (2 of 2 overloads)

Inherited from basic_socket.

Get the local endpoint of the socket.

```
endpoint_type local_endpoint(
    boost::system::error_code & ec) const;
```

This function is used to obtain the locally bound endpoint of the socket.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

An object that represents the local endpoint of the socket. Returns a default-constructed endpoint object if an error occurred.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
boost::asio::ip::tcp::endpoint endpoint = socket.local_endpoint(ec);
if (ec)
{
  // An error occurred.
}
```

basic_datagram_socket::lowest_layer

Get a reference to the lowest layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

basic_datagram_socket::lowest_layer (1 of 2 overloads)

Inherited from basic_socket.

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

This function returns a reference to the lowest layer in a stack of layers. Since a `basic_socket` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

basic_datagram_socket::lowest_layer (2 of 2 overloads)

Inherited from basic_socket.

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

This function returns a const reference to the lowest layer in a stack of layers. Since a `basic_socket` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A const reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

basic_datagram_socket::lowest_layer_type

Inherited from basic_socket.

A `basic_socket` is always the lowest layer.

```
typedef basic_socket< Protocol, DatagramSocketService > lowest_layer_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_socket	Construct a basic_socket without opening it. Construct and open a basic_socket. Construct a basic_socket, opening it and binding it to the given local endpoint. Construct a basic_socket on an existing native socket. Move-construct a basic_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_socket from another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
remote_endpoint	Get the remote endpoint of the socket.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_socket	Protected destructor to prevent deletion through this type.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_socket` class template provides functionality that is common to both stream-oriented and datagram-oriented sockets.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_datagram_socket::max_connections

Inherited from socket_base.

The maximum length of the queue of pending incoming connections.

```
static const int max_connections = implementation_defined;
```

basic_datagram_socket::message_do_not_route

Inherited from socket_base.

Specify that the data should not be subject to routing.

```
static const int message_do_not_route = implementation_defined;
```

basic_datagram_socket::message_end_of_record

Inherited from socket_base.

Specifies that the data marks the end of a record.

```
static const int message_end_of_record = implementation_defined;
```

basic_datagram_socket::message_flags

Inherited from socket_base.

Bitmask type for flags that can be passed to send and receive operations.

```
typedef int message_flags;
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::message_out_of_band

Inherited from socket_base.

Process out-of-band data.

```
static const int message_out_of_band = implementation_defined;
```

basic_datagram_socket::message_peek

Inherited from socket_base.

Peek at incoming data without removing it from the input queue.

```
static const int message_peek = implementation_defined;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_datagram_socket::native

Inherited from basic_socket.

(Deprecated: Use `native_handle()`.) Get the native socket representation.

```
native_type native();
```

This function may be used to obtain the underlying representation of the socket. This is intended to allow access to native socket functionality that is not otherwise provided.

basic_datagram_socket::native_handle

Inherited from basic_socket.

Get the native socket representation.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying representation of the socket. This is intended to allow access to native socket functionality that is not otherwise provided.

basic_datagram_socket::native_handle_type

The native representation of a socket.

```
typedef DatagramSocketService::native_handle_type native_handle_type;
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::native_non_blocking

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking() const;
  » more...
```

Sets the non-blocking mode of the native socket implementation.

```
void native_non_blocking(
    bool mode);
  » more...

boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::native_non_blocking (1 of 3 overloads)

Inherited from basic_socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking() const;
```

This function is used to retrieve the non-blocking mode of the underlying native socket. This mode has no effect on the behaviour of the socket object's synchronous operations.

Return Value

`true` if the underlying socket is in non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Remarks

The current non-blocking mode is cached by the socket object. Consequently, the return value may be incorrect if the non-blocking mode was set directly on the native socket.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_datagram_socket::native_non_blocking (2 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the native socket implementation.

```
void native_non_blocking(
    bool mode);
```

This function is used to modify the non-blocking mode of the underlying native socket. It has no effect on the behaviour of the socket object's synchronous operations.

Parameters

mode — If `true`, the underlying socket is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Exceptions

boost::system::system_error — Thrown on failure. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_datagram_socket::native_non_blocking (3 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the native socket implementation.

```
boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

This function is used to modify the non-blocking mode of the underlying native socket. It has no effect on the behaviour of the socket object's synchronous operations.

Parameters

mode — If `true`, the underlying socket is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

ec — Set to indicate what error occurred, if any. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_datagram_socket::native_type

(Deprecated: Use native_handle_type.) The native representation of a socket.

```
typedef DatagramSocketService::native_handle_type native_type;
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::non_blocking

Gets the non-blocking mode of the socket.

```
bool non_blocking() const;
  » more...
```

Sets the non-blocking mode of the socket.

```
void non_blocking(
    bool mode);
  » more...

boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::non_blocking (1 of 3 overloads)

Inherited from basic_socket.

Gets the non-blocking mode of the socket.

```
bool non_blocking() const;
```

Return Value

`true` if the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_datagram_socket::non_blocking (2 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the socket.

```
void non_blocking(
    bool mode);
```

Parameters

mode — If `true`, the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Exceptions

boost::system::system_error — Thrown on failure.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_datagram_socket::non_blocking (3 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the socket.

```
boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

Parameters

mode — If `true`, the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

ec — Set to indicate what error occurred, if any.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_datagram_socket::non_blocking_io

Inherited from socket_base.

(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.

```
typedef implementation_defined non_blocking_io;
```

Implements the FIONBIO IO control command.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::non_blocking_io command(true);
socket.io_control(command);
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::open

Open the socket using the specified protocol.

```
void open(
    const protocol_type & protocol = protocol_type());
  » more...

boost::system::error_code open(
    const protocol_type & protocol,
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::open (1 of 2 overloads)

Inherited from basic_socket.

Open the socket using the specified protocol.

```
void open(
    const protocol_type & protocol = protocol_type());
```

This function opens the socket so that it will use the specified protocol.

Parameters

protocol An object specifying protocol parameters to be used.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
```

basic_datagram_socket::open (2 of 2 overloads)

Inherited from basic_socket.

Open the socket using the specified protocol.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code open(
    const protocol_type & protocol,
    boost::system::error_code & ec);
```

This function opens the socket so that it will use the specified protocol.

Parameters

protocol — An object specifying which protocol is to be used.

ec — Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::system::error_code ec;
socket.open(boost::asio::ip::tcp::v4(), ec);
if (ec)
{
  // An error occurred.
}
```

basic_datagram_socket::operator=

Move-assign a `basic_datagram_socket` from another.

```
basic_datagram_socket & operator=(
    basic_datagram_socket && other);
```

This assignment operator moves a datagram socket from one object to another.

Parameters

other — The other `basic_datagram_socket` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_datagram_socket(io_service&)` `constructor`.

basic_datagram_socket::protocol_type

The protocol type.

```
typedef Protocol protocol_type;
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::receive

Receive some data on a connected socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers);
  » more...

template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags);
  » more...

template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::receive (1 of 3 overloads)

Receive some data on a connected socket.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers);
```

This function is used to receive data on the datagram socket. The function call will block until data has been received successfully or an error occurs.

Parameters

buffers One or more buffers into which the data will be received.

Return Value

The number of bytes received.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

The receive operation can only be used with a connected socket. Use the receive_from function to receive data on an unconnected datagram socket.

Example

To receive into a single data buffer use the `buffer` function as follows:

```
socket.receive(boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_datagram_socket::receive (2 of 3 overloads)

Receive some data on a connected socket.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags);
```

This function is used to receive data on the datagram socket. The function call will block until data has been received successfully or an error occurs.

Parameters

buffers One or more buffers into which the data will be received.

flags Flags specifying how the receive call is to be made.

Return Value

The number of bytes received.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

The receive operation can only be used with a connected socket. Use the receive_from function to receive data on an unconnected datagram socket.

basic_datagram_socket::receive (3 of 3 overloads)

Receive some data on a connected socket.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

This function is used to receive data on the datagram socket. The function call will block until data has been received successfully or an error occurs.

Parameters

buffers One or more buffers into which the data will be received.

flags Flags specifying how the receive call is to be made.

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes received.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

The receive operation can only be used with a connected socket. Use the receive_from function to receive data on an unconnected datagram socket.

basic_datagram_socket::receive_buffer_size

Inherited from socket_base.

Socket option for the receive buffer size of a socket.

```
typedef implementation_defined receive_buffer_size;
```

Implements the SOL_SOCKET/SO_RCVBUF socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::receive_from

Receive a datagram with the endpoint of the sender.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint);
  » more...

template<
    typename MutableBufferSequence>
std::size_t receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags);
  » more...

template<
    typename MutableBufferSequence>
std::size_t receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::receive_from (1 of 3 overloads)

Receive a datagram with the endpoint of the sender.

```
template<
    typename MutableBufferSequence>
std::size_t receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint);
```

This function is used to receive a datagram. The function call will block until data has been received successfully or an error occurs.

Parameters

buffers	One or more buffers into which the data will be received.
sender_endpoint	An endpoint object that receives the endpoint of the remote sender of the datagram.

Return Value

The number of bytes received.

Exceptions

boost::system::system_error	Thrown on failure.

Example

To receive into a single data buffer use the `buffer` function as follows:

```
boost::asio::ip::udp::endpoint sender_endpoint;
socket.receive_from(
    boost::asio::buffer(data, size), sender_endpoint);
```

See the `buffer` documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_datagram_socket::receive_from (2 of 3 overloads)

Receive a datagram with the endpoint of the sender.

```
template<
    typename MutableBufferSequence>
std::size_t receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags);
```

This function is used to receive a datagram. The function call will block until data has been received successfully or an error occurs.

Parameters

buffers	One or more buffers into which the data will be received.
sender_endpoint	An endpoint object that receives the endpoint of the remote sender of the datagram.
flags	Flags specifying how the receive call is to be made.

Return Value

The number of bytes received.

Exceptions

boost::system::system_error	Thrown on failure.

basic_datagram_socket::receive_from (3 of 3 overloads)

Receive a datagram with the endpoint of the sender.

```
template<
    typename MutableBufferSequence>
std::size_t receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

This function is used to receive a datagram. The function call will block until data has been received successfully or an error occurs.

Parameters

buffers	One or more buffers into which the data will be received.
sender_endpoint	An endpoint object that receives the endpoint of the remote sender of the datagram.
flags	Flags specifying how the receive call is to be made.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes received.

basic_datagram_socket::receive_low_watermark

Inherited from socket_base.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Socket option for the receive low watermark.

```
typedef implementation_defined receive_low_watermark;
```

Implements the SOL_SOCKET/SO_RCVLOWAT socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::remote_endpoint

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint() const;
  » more...

endpoint_type remote_endpoint(
    boost::system::error_code & ec) const;
  » more...
```

basic_datagram_socket::remote_endpoint (1 of 2 overloads)

Inherited from basic_socket.

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint() const;
```

This function is used to obtain the remote endpoint of the socket.

Return Value

An object that represents the remote endpoint of the socket.

Exceptions

boost::system::system_error	Thrown on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::endpoint endpoint = socket.remote_endpoint();
```

basic_datagram_socket::remote_endpoint (2 of 2 overloads)

Inherited from basic_socket.

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint(
    boost::system::error_code & ec) const;
```

This function is used to obtain the remote endpoint of the socket.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

An object that represents the remote endpoint of the socket. Returns a default-constructed endpoint object if an error occurred.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
boost::asio::ip::tcp::endpoint endpoint = socket.remote_endpoint(ec);
if (ec)
{
  // An error occurred.
}
```

basic_datagram_socket::reuse_address

Inherited from socket_base.

Socket option to allow the socket to be bound to an address that is already in use.

```
typedef implementation_defined reuse_address;
```

Implements the SOL_SOCKET/SO_REUSEADDR socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option(true);
acceptor.set_option(option);
```

Getting the current option value:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::send

Send some data on a connected socket.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers);
  » more...

template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags);
  » more...

template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::send (1 of 3 overloads)

Send some data on a connected socket.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers);
```

This function is used to send data on the datagram socket. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers One ore more data buffers to be sent on the socket.

Return Value

The number of bytes sent.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

The send operation can only be used with a connected socket. Use the send_to function to send data on an unconnected datagram socket.

Example

To send a single data buffer use the `buffer` function as follows:

```
socket.send(boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on sending multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_datagram_socket::send (2 of 3 overloads)

Send some data on a connected socket.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags);
```

This function is used to send data on the datagram socket. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers	One ore more data buffers to be sent on the socket.
flags	Flags specifying how the send call is to be made.

Return Value

The number of bytes sent.

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

The send operation can only be used with a connected socket. Use the send_to function to send data on an unconnected datagram socket.

basic_datagram_socket::send (3 of 3 overloads)

Send some data on a connected socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

This function is used to send data on the datagram socket. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers	One or more data buffers to be sent on the socket.
flags	Flags specifying how the send call is to be made.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes sent.

Remarks

The send operation can only be used with a connected socket. Use the send_to function to send data on an unconnected datagram socket.

basic_datagram_socket::send_buffer_size

Inherited from socket_base.

Socket option for the send buffer size of a socket.

```
typedef implementation_defined send_buffer_size;
```

Implements the SOL_SOCKET/SO_SNDBUF socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio.hpp`

basic_datagram_socket::send_low_watermark

Inherited from socket_base.

Socket option for the send low watermark.

```
typedef implementation_defined send_low_watermark;
```

Implements the SOL_SOCKET/SO_SNDLOWAT socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::send_to

Send a datagram to the specified endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination);
  » more...

template<
    typename ConstBufferSequence>
std::size_t send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags);
  » more...

template<
    typename ConstBufferSequence>
std::size_t send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::send_to (1 of 3 overloads)

Send a datagram to the specified endpoint.

```
template<
    typename ConstBufferSequence>
std::size_t send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination);
```

This function is used to send a datagram to the specified remote endpoint. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers	One or more data buffers to be sent to the remote endpoint.
destination	The remote endpoint to which the data will be sent.

Return Value

The number of bytes sent.

Exceptions

boost::system::system_error	Thrown on failure.

Example

To send a single data buffer use the `buffer` function as follows:

```
boost::asio::ip::udp::endpoint destination(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.send_to(boost::asio::buffer(data, size), destination);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

See the buffer documentation for information on sending multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_datagram_socket::send_to (2 of 3 overloads)

Send a datagram to the specified endpoint.

```
template<
    typename ConstBufferSequence>
std::size_t send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags);
```

This function is used to send a datagram to the specified remote endpoint. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers	One or more data buffers to be sent to the remote endpoint.
destination	The remote endpoint to which the data will be sent.
flags	Flags specifying how the send call is to be made.

Return Value

The number of bytes sent.

Exceptions

boost::system::system_error	Thrown on failure.

basic_datagram_socket::send_to (3 of 3 overloads)

Send a datagram to the specified endpoint.

```
template<
    typename ConstBufferSequence>
std::size_t send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

This function is used to send a datagram to the specified remote endpoint. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers	One or more data buffers to be sent to the remote endpoint.
destination	The remote endpoint to which the data will be sent.
flags	Flags specifying how the send call is to be made.
ec	Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Return Value

The number of bytes sent.

basic_datagram_socket::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

basic_datagram_socket::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef DatagramSocketService service_type;
```

Requirements

Header: `boost/asio/basic_datagram_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_datagram_socket::set_option

Set an option on the socket.

```
void set_option(
    const SettableSocketOption & option);
  » more...

boost::system::error_code set_option(
    const SettableSocketOption & option,
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::set_option (1 of 2 overloads)

Inherited from basic_socket.

Set an option on the socket.

```
template<
    typename SettableSocketOption>
void set_option(
    const SettableSocketOption & option);
```

This function is used to set an option on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

option The new option value to be set on the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Setting the IPPROTO_TCP/TCP_NODELAY option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::no_delay option(true);
socket.set_option(option);
```

basic_datagram_socket::set_option (2 of 2 overloads)

Inherited from basic_socket.

Set an option on the socket.

```
template<
    typename SettableSocketOption>
boost::system::error_code set_option(
    const SettableSocketOption & option,
    boost::system::error_code & ec);
```

This function is used to set an option on the socket.

Parameters

option The new option value to be set on the socket.

ec Set to indicate what error occurred, if any.

Example

Setting the IPPROTO_TCP/TCP_NODELAY option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::no_delay option(true);
boost::system::error_code ec;
socket.set_option(option, ec);
if (ec)
{
  // An error occurred.
}
```

basic_datagram_socket::shutdown

Disable sends or receives on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void shutdown(
    shutdown_type what);
  » more...

boost::system::error_code shutdown(
    shutdown_type what,
    boost::system::error_code & ec);
  » more...
```

basic_datagram_socket::shutdown (1 of 2 overloads)

Inherited from basic_socket.

Disable sends or receives on the socket.

```
void shutdown(
    shutdown_type what);
```

This function is used to disable send operations, receive operations, or both.

Parameters

what Determines what types of operation will no longer be allowed.

Exceptions

boost::system::system_error Thrown on failure.

Example

Shutting down the send side of the socket:

```
boost::asio::ip::tcp::socket socket(io_service);
...
socket.shutdown(boost::asio::ip::tcp::socket::shutdown_send);
```

basic_datagram_socket::shutdown (2 of 2 overloads)

Inherited from basic_socket.

Disable sends or receives on the socket.

```
boost::system::error_code shutdown(
    shutdown_type what,
    boost::system::error_code & ec);
```

This function is used to disable send operations, receive operations, or both.

Parameters

what Determines what types of operation will no longer be allowed.

ec Set to indicate what error occurred, if any.

Example

Shutting down the send side of the socket:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
socket.shutdown(boost::asio::ip::tcp::socket::shutdown_send, ec);
if (ec)
{
  // An error occurred.
}
```

basic_datagram_socket::shutdown_type

Inherited from socket_base.

Different ways a socket may be shutdown.

```
enum shutdown_type
```

Values

shutdown_receive	Shutdown the receive side of the socket.
shutdown_send	Shutdown the send side of the socket.
shutdown_both	Shutdown both send and receive on the socket.

basic_deadline_timer

Provides waitable timer functionality.

```
template<
    typename Time,
    typename TimeTraits = boost::asio::time_traits<Time>,
    typename TimerService = deadline_timer_service<Time, TimeTraits>>
class basic_deadline_timer :
  public basic_io_object< TimerService >
```

Types

Name	Description
duration_type	The duration type.
implementation_type	The underlying implementation type of I/O object.
service_type	The type of the service that will be used to provide I/O operations.
time_type	The time type.
traits_type	The time traits type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
async_wait	Start an asynchronous wait on the timer.
basic_deadline_timer	Constructor. Constructor to set a particular expiry time as an absolute time. Constructor to set a particular expiry time relative to now.
cancel	Cancel any asynchronous operations that are waiting on the timer.
cancel_one	Cancels one asynchronous operation that is waiting on the timer.
expires_at	Get the timer's expiry time as an absolute time. Set the timer's expiry time as an absolute time.
expires_from_now	Get the timer's expiry time relative to now. Set the timer's expiry time relative to now.
get_io_service	Get the io_service associated with the object.
wait	Perform a blocking wait on the timer.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_deadline_timer` class template provides the ability to perform a blocking or asynchronous wait for a timer to expire.

A deadline timer is always in one of two states: "expired" or "not expired". If the `wait()` or `async_wait()` function is called on an expired timer, the wait operation will complete immediately.

Most applications will use the `deadline_timer` typedef.

Thread Safety

Distinct objects: Safe.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Shared objects: Unsafe.

Examples

Performing a blocking wait:

```
// Construct a timer without setting an expiry time.
boost::asio::deadline_timer timer(io_service);

// Set an expiry time relative to now.
timer.expires_from_now(boost::posix_time::seconds(5));

// Wait for the timer to expire.
timer.wait();
```

Performing an asynchronous wait:

```
void handler(const boost::system::error_code& error)
{
  if (!error)
  {
    // Timer expired.
  }
}

...

// Construct a timer with an absolute expiry time.
boost::asio::deadline_timer timer(io_service,
    boost::posix_time::time_from_string("2005-12-07 23:59:59.000"));

// Start an asynchronous wait.
timer.async_wait(handler);
```

Changing an active deadline_timer's expiry time

Changing the expiry time of a timer while there are pending asynchronous waits causes those wait operations to be cancelled. To ensure that the action associated with the timer is performed only once, use something like this: used:

```
void on_some_event()
{
  if (my_timer.expires_from_now(seconds(5)) > 0)
  {
    // We managed to cancel the timer. Start new asynchronous wait.
    my_timer.async_wait(on_timeout);
  }
  else
  {
    // Too late, timer has already expired!
  }
}

void on_timeout(const boost::system::error_code& e)
{
  if (e != boost::asio::error::operation_aborted)
  {
    // Timer was not cancelled, take necessary action.
  }
}
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- The `boost::asio::basic_deadline_timer::expires_from_now()` function cancels any pending asynchronous waits, and returns the number of asynchronous waits that were cancelled. If it returns 0 then you were too late and the wait handler has already been executed, or will soon be executed. If it returns 1 then the wait handler was successfully cancelled.
- If a wait handler is cancelled, the boost::system::error_code passed to it contains the value `boost::asio::error::operation_aborted`.

Requirements

Header: `boost/asio/basic_deadline_timer.hpp`

Convenience header: `boost/asio.hpp`

basic_deadline_timer::async_wait

Start an asynchronous wait on the timer.

```
template<
    typename WaitHandler>
void async_wait(
    WaitHandler handler);
```

This function may be used to initiate an asynchronous wait against the timer. It always returns immediately.

For each call to `async_wait()`, the supplied handler will be called exactly once. The handler will be called when:

- The timer has expired.
- The timer was cancelled, in which case the handler is passed the error code `boost::asio::error::operation_aborted`.

Parameters

handler — The handler to be called when the timer expires. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error // Result of operation.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

basic_deadline_timer::basic_deadline_timer

Constructor.

```
explicit basic_deadline_timer(
    boost::asio::io_service & io_service);
  » more...
```

Constructor to set a particular expiry time as an absolute time.

```
basic_deadline_timer(
    boost::asio::io_service & io_service,
    const time_type & expiry_time);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Constructor to set a particular expiry time relative to now.

```
basic_deadline_timer(
    boost::asio::io_service & io_service,
    const duration_type & expiry_time);
  » more...
```

basic_deadline_timer::basic_deadline_timer (1 of 3 overloads)

Constructor.

```
basic_deadline_timer(
    boost::asio::io_service & io_service);
```

This constructor creates a timer without setting an expiry time. The `expires_at()` or `expires_from_now()` functions must be called to set an expiry time before the timer can be waited on.

Parameters

io_service — The `io_service` object that the timer will use to dispatch handlers for any asynchronous operations performed on the timer.

basic_deadline_timer::basic_deadline_timer (2 of 3 overloads)

Constructor to set a particular expiry time as an absolute time.

```
basic_deadline_timer(
    boost::asio::io_service & io_service,
    const time_type & expiry_time);
```

This constructor creates a timer and sets the expiry time.

Parameters

io_service — The `io_service` object that the timer will use to dispatch handlers for any asynchronous operations performed on the timer.

expiry_time — The expiry time to be used for the timer, expressed as an absolute time.

basic_deadline_timer::basic_deadline_timer (3 of 3 overloads)

Constructor to set a particular expiry time relative to now.

```
basic_deadline_timer(
    boost::asio::io_service & io_service,
    const duration_type & expiry_time);
```

This constructor creates a timer and sets the expiry time.

Parameters

io_service — The `io_service` object that the timer will use to dispatch handlers for any asynchronous operations performed on the timer.

expiry_time — The expiry time to be used for the timer, relative to now.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_deadline_timer::cancel

Cancel any asynchronous operations that are waiting on the timer.

```
std::size_t cancel();
  » more...

std::size_t cancel(
    boost::system::error_code & ec);
  » more...
```

basic_deadline_timer::cancel (1 of 2 overloads)

Cancel any asynchronous operations that are waiting on the timer.

```
std::size_t cancel();
```

This function forces the completion of any pending asynchronous wait operations against the timer. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Cancelling the timer does not change the expiry time.

Return Value

The number of asynchronous operations that were cancelled.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

If the timer has already expired when `cancel()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_deadline_timer::cancel (2 of 2 overloads)

Cancel any asynchronous operations that are waiting on the timer.

```
std::size_t cancel(
    boost::system::error_code & ec);
```

This function forces the completion of any pending asynchronous wait operations against the timer. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Cancelling the timer does not change the expiry time.

Parameters

ec Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Return Value

The number of asynchronous operations that were cancelled.

Remarks

If the timer has already expired when `cancel()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_deadline_timer::cancel_one

Cancels one asynchronous operation that is waiting on the timer.

```
std::size_t cancel_one();
  » more...

std::size_t cancel_one(
    boost::system::error_code & ec);
  » more...
```

basic_deadline_timer::cancel_one (1 of 2 overloads)

Cancels one asynchronous operation that is waiting on the timer.

```
std::size_t cancel_one();
```

This function forces the completion of one pending asynchronous wait operation against the timer. Handlers are cancelled in FIFO order. The handler for the cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Cancelling the timer does not change the expiry time.

Return Value

The number of asynchronous operations that were cancelled. That is, either 0 or 1.

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

If the timer has already expired when `cancel_one()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_deadline_timer::cancel_one (2 of 2 overloads)

Cancels one asynchronous operation that is waiting on the timer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t cancel_one(
    boost::system::error_code & ec);
```

This function forces the completion of one pending asynchronous wait operation against the timer. Handlers are cancelled in FIFO order. The handler for the cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Cancelling the timer does not change the expiry time.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

The number of asynchronous operations that were cancelled. That is, either 0 or 1.

Remarks

If the timer has already expired when `cancel_one()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_deadline_timer::duration_type

The duration type.

```
typedef traits_type::duration_type duration_type;
```

Requirements

Header: `boost/asio/basic_deadline_timer.hpp`

Convenience header: `boost/asio.hpp`

basic_deadline_timer::expires_at

Get the timer's expiry time as an absolute time.

```
time_type expires_at() const;
  » more...
```

Set the timer's expiry time as an absolute time.

```
std::size_t expires_at(
    const time_type & expiry_time);
  » more...

std::size_t expires_at(
    const time_type & expiry_time,
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_deadline_timer::expires_at (1 of 3 overloads)

Get the timer's expiry time as an absolute time.

```
time_type expires_at() const;
```

This function may be used to obtain the timer's current expiry time. Whether the timer has expired or not does not affect this value.

basic_deadline_timer::expires_at (2 of 3 overloads)

Set the timer's expiry time as an absolute time.

```
std::size_t expires_at(
    const time_type & expiry_time);
```

This function sets the expiry time. Any pending asynchronous wait operations will be cancelled. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Parameters

expiry_time	The expiry time to be used for the timer.

Return Value

The number of asynchronous operations that were cancelled.

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

If the timer has already expired when `expires_at()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_deadline_timer::expires_at (3 of 3 overloads)

Set the timer's expiry time as an absolute time.

```
std::size_t expires_at(
    const time_type & expiry_time,
    boost::system::error_code & ec);
```

This function sets the expiry time. Any pending asynchronous wait operations will be cancelled. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Parameters

expiry_time	The expiry time to be used for the timer.
ec	Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Return Value

The number of asynchronous operations that were cancelled.

Remarks

If the timer has already expired when `expires_at()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_deadline_timer::expires_from_now

Get the timer's expiry time relative to now.

```
duration_type expires_from_now() const;
  » more...
```

Set the timer's expiry time relative to now.

```
std::size_t expires_from_now(
    const duration_type & expiry_time);
  » more...

std::size_t expires_from_now(
    const duration_type & expiry_time,
    boost::system::error_code & ec);
  » more...
```

basic_deadline_timer::expires_from_now (1 of 3 overloads)

Get the timer's expiry time relative to now.

```
duration_type expires_from_now() const;
```

This function may be used to obtain the timer's current expiry time. Whether the timer has expired or not does not affect this value.

basic_deadline_timer::expires_from_now (2 of 3 overloads)

Set the timer's expiry time relative to now.

```
std::size_t expires_from_now(
    const duration_type & expiry_time);
```

This function sets the expiry time. Any pending asynchronous wait operations will be cancelled. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Parameters

expiry_time	The expiry time to be used for the timer.

Return Value

The number of asynchronous operations that were cancelled.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

If the timer has already expired when `expires_from_now()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_deadline_timer::expires_from_now (3 of 3 overloads)

Set the timer's expiry time relative to now.

```
std::size_t expires_from_now(
    const duration_type & expiry_time,
    boost::system::error_code & ec);
```

This function sets the expiry time. Any pending asynchronous wait operations will be cancelled. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Parameters

expiry_time	The expiry time to be used for the timer.
ec	Set to indicate what error occurred, if any.

Return Value

The number of asynchronous operations that were cancelled.

Remarks

If the timer has already expired when `expires_from_now()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_deadline_timer::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

basic_deadline_timer::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

basic_deadline_timer::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

basic_deadline_timer::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

basic_deadline_timer::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

basic_deadline_timer::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

basic_deadline_timer::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

basic_deadline_timer::implementation

Inherited from basic_io_object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

basic_deadline_timer::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/basic_deadline_timer.hpp`

Convenience header: `boost/asio.hpp`

basic_deadline_timer::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

basic_deadline_timer::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef TimerService service_type;
```

Requirements

Header: `boost/asio/basic_deadline_timer.hpp`

Convenience header: `boost/asio.hpp`

basic_deadline_timer::time_type

The time type.

```
typedef traits_type::time_type time_type;
```

Requirements

Header: `boost/asio/basic_deadline_timer.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_deadline_timer::traits_type

The time traits type.

```
typedef TimeTraits traits_type;
```

Requirements

Header: `boost/asio/basic_deadline_timer.hpp`

Convenience header: `boost/asio.hpp`

basic_deadline_timer::wait

Perform a blocking wait on the timer.

```
void wait();
  » more...

void wait(
    boost::system::error_code & ec);
  » more...
```

basic_deadline_timer::wait (1 of 2 overloads)

Perform a blocking wait on the timer.

```
void wait();
```

This function is used to wait for the timer to expire. This function blocks and does not return until the timer has expired.

Exceptions

boost::system::system_error	Thrown on failure.

basic_deadline_timer::wait (2 of 2 overloads)

Perform a blocking wait on the timer.

```
void wait(
    boost::system::error_code & ec);
```

This function is used to wait for the timer to expire. This function blocks and does not return until the timer has expired.

Parameters

ec	Set to indicate what error occurred, if any.

basic_io_object

Base class for all I/O objects.

```
template<
    typename IoObjectService>
class basic_io_object
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
service_type	The type of the service that will be used to provide I/O operations.

Member Functions

Name	Description
get_io_service	Get the io_service associated with the object.

Protected Member Functions

Name	Description
basic_io_object	Construct a basic_io_object. Move-construct a basic_io_object.
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
operator=	Move-assign a basic_io_object.
~basic_io_object	Protected destructor to prevent deletion through this type.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

Remarks

All I/O objects are non-copyable. However, when using C++0x, certain I/O objects do support move construction and move assignment.

Requirements

Header: `boost/asio/basic_io_object.hpp`

Convenience header: `boost/asio.hpp`

basic_io_object::basic_io_object

Construct a `basic_io_object`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
explicit basic_io_object(
    boost::asio::io_service & io_service);
  » more...
```

Move-construct a basic_io_object.

```
basic_io_object(
    basic_io_object && other);
  » more...
```

basic_io_object::basic_io_object (1 of 2 overloads)

Construct a basic_io_object.

```
basic_io_object(
    boost::asio::io_service & io_service);
```

Performs:

```
get_service().construct(get_implementation());
```

basic_io_object::basic_io_object (2 of 2 overloads)

Move-construct a basic_io_object.

```
basic_io_object(
    basic_io_object && other);
```

Performs:

```
get_service().move_construct(
    get_implementation(), other.get_implementation());
```

Remarks

Available only for services that support movability,

basic_io_object::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

basic_io_object::get_implementation (1 of 2 overloads)

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_io_object::get_implementation (2 of 2 overloads)

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

basic_io_object::get_io_service

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

basic_io_object::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

basic_io_object::get_service (1 of 2 overloads)

Get the service associated with the I/O object.

```
service_type & get_service();
```

basic_io_object::get_service (2 of 2 overloads)

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

basic_io_object::implementation

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

basic_io_object::implementation_type

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/basic_io_object.hpp`

Convenience header: `boost/asio.hpp`

basic_io_object::operator=

Move-assign a `basic_io_object`.

```
basic_io_object & operator=(
    basic_io_object && other);
```

Performs:

```
get_service().move_assign(get_implementation(),
    other.get_service(), other.get_implementation());
```

Remarks

Available only for services that support movability,

basic_io_object::service

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

basic_io_object::service_type

The type of the service that will be used to provide I/O operations.

```
typedef IoObjectService service_type;
```

Requirements

Header: `boost/asio/basic_io_object.hpp`

Convenience header: `boost/asio.hpp`

basic_io_object::~basic_io_object

Protected destructor to prevent deletion through this type.

```
~basic_io_object();
```

Performs:

```
get_service().destroy(get_implementation());
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket

Provides raw-oriented socket functionality.

```
template<
    typename Protocol,
    typename RawSocketService = raw_socket_service<Protocol>>
class basic_raw_socket :
  public basic_socket< Protocol, RawSocketService >
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
async_receive	Start an asynchronous receive on a connected socket.
async_receive_from	Start an asynchronous receive.
async_send	Start an asynchronous send on a connected socket.
async_send_to	Start an asynchronous send.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_raw_socket	Construct a basic_raw_socket without opening it. Construct and open a basic_raw_socket. Construct a basic_raw_socket, opening it and binding it to the given local endpoint. Construct a basic_raw_socket on an existing native socket. Move-construct a basic_raw_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_raw_socket from another.
receive	Receive some data on a connected socket.
receive_from	Receive raw data with the endpoint of the sender.
remote_endpoint	Get the remote endpoint of the socket.
send	Send some data on a connected socket.
send_to	Send raw data to the specified endpoint.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_raw_socket` class template provides asynchronous and blocking raw-oriented socket functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::assign

Assign an existing native socket to the socket.

```
void assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket);
  » more...

boost::system::error_code assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::assign (1 of 2 overloads)

Inherited from basic_socket.

Assign an existing native socket to the socket.

```
void assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket);
```

basic_raw_socket::assign (2 of 2 overloads)

Inherited from basic_socket.

Assign an existing native socket to the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
```

basic_raw_socket::async_connect

Inherited from basic_socket.

Start an asynchronous connect.

```
template<
    typename ConnectHandler>
void async_connect(
    const endpoint_type & peer_endpoint,
    ConnectHandler handler);
```

This function is used to asynchronously connect a socket to the specified remote endpoint. The function call always returns immediately.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint — The remote endpoint to which the socket will be connected. Copies will be made of the endpoint object as required.

handler — The handler to be called when the connection operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error // Result of operation
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

```
void connect_handler(const boost::system::error_code& error)
{
  if (!error)
  {
    // Connect succeeded.
  }
}

...

boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.async_connect(endpoint, connect_handler);
```

basic_raw_socket::async_receive

Start an asynchronous receive on a connected socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    ReadHandler handler);
  » more...

template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    ReadHandler handler);
  » more...
```

basic_raw_socket::async_receive (1 of 2 overloads)

Start an asynchronous receive on a connected socket.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

This function is used to asynchronously receive data from the raw socket. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be received. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the receive operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes received.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The async_receive operation can only be used with a connected socket. Use the async_receive_from function to receive data on an unconnected raw socket.

Example

To receive into a single data buffer use the `buffer` function as follows:

```
socket.async_receive(boost::asio::buffer(data, size), handler);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

See the buffer documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_raw_socket::async_receive (2 of 2 overloads)

Start an asynchronous receive on a connected socket.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    ReadHandler handler);
```

This function is used to asynchronously receive data from the raw socket. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be received. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

flags — Flags specifying how the receive call is to be made.

handler — The handler to be called when the receive operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes received.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The async_receive operation can only be used with a connected socket. Use the async_receive_from function to receive data on an unconnected raw socket.

basic_raw_socket::async_receive_from

Start an asynchronous receive.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    ReadHandler handler);
  » more...

template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags,
    ReadHandler handler);
  » more...
```

basic_raw_socket::async_receive_from (1 of 2 overloads)

Start an asynchronous receive.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    ReadHandler handler);
```

This function is used to asynchronously receive raw data. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be received. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

sender_endpoint — An endpoint object that receives the endpoint of the remote sender of the data. Ownership of the sender_endpoint object is retained by the caller, which must guarantee that it is valid until the handler is called.

handler — The handler to be called when the receive operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes received.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To receive into a single data buffer use the `buffer` function as follows:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
socket.async_receive_from(
    boost::asio::buffer(data, size), 0, sender_endpoint, handler);
```

See the buffer documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_raw_socket::async_receive_from (2 of 2 overloads)

Start an asynchronous receive.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags,
    ReadHandler handler);
```

This function is used to asynchronously receive raw data. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be received. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

sender_endpoint — An endpoint object that receives the endpoint of the remote sender of the data. Ownership of the sender_endpoint object is retained by the caller, which must guarantee that it is valid until the handler is called.

flags — Flags specifying how the receive call is to be made.

handler — The handler to be called when the receive operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes received.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

basic_raw_socket::async_send

Start an asynchronous send on a connected socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    const ConstBufferSequence & buffers,
    WriteHandler handler);
  » more...

template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    WriteHandler handler);
  » more...
```

basic_raw_socket::async_send (1 of 2 overloads)

Start an asynchronous send on a connected socket.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

This function is used to send data on the raw socket. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers — One or more data buffers to be sent on the socket. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the send operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes sent.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The async_send operation can only be used with a connected socket. Use the async_send_to function to send data on an unconnected raw socket.

Example

To send a single data buffer use the `buffer` function as follows:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
socket.async_send(boost::asio::buffer(data, size), handler);
```

See the buffer documentation for information on sending multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_raw_socket::async_send (2 of 2 overloads)

Start an asynchronous send on a connected socket.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    WriteHandler handler);
```

This function is used to send data on the raw socket. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers — One or more data buffers to be sent on the socket. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

flags — Flags specifying how the send call is to be made.

handler — The handler to be called when the send operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes sent.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The async_send operation can only be used with a connected socket. Use the async_send_to function to send data on an unconnected raw socket.

basic_raw_socket::async_send_to

Start an asynchronous send.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    WriteHandler handler);
  » more...

template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags,
    WriteHandler handler);
  » more...
```

basic_raw_socket::async_send_to (1 of 2 overloads)

Start an asynchronous send.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    WriteHandler handler);
```

This function is used to asynchronously send raw data to the specified remote endpoint. The function call always returns immediately.

Parameters

buffers — One or more data buffers to be sent to the remote endpoint. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

destination — The remote endpoint to which the data will be sent. Copies will be made of the endpoint as required.

handler — The handler to be called when the send operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes sent.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To send a single data buffer use the `buffer` function as follows:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::udp::endpoint destination(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.async_send_to(
    boost::asio::buffer(data, size), destination, handler);
```

See the `buffer` documentation for information on sending multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_raw_socket::async_send_to (2 of 2 overloads)

Start an asynchronous send.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags,
    WriteHandler handler);
```

This function is used to asynchronously send raw data to the specified remote endpoint. The function call always returns immediately.

Parameters

buffers — One or more data buffers to be sent to the remote endpoint. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

flags — Flags specifying how the send call is to be made.

destination — The remote endpoint to which the data will be sent. Copies will be made of the endpoint as required.

handler — The handler to be called when the send operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes sent.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

basic_raw_socket::at_mark

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark() const;
  » more...

bool at_mark(
    boost::system::error_code & ec) const;
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket::at_mark (1 of 2 overloads)

Inherited from basic_socket.

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark() const;
```

This function is used to check whether the socket input is currently positioned at the out-of-band data mark.

Return Value

A bool indicating whether the socket is at the out-of-band data mark.

Exceptions

boost::system::system_error Thrown on failure.

basic_raw_socket::at_mark (2 of 2 overloads)

Inherited from basic_socket.

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark(
    boost::system::error_code & ec) const;
```

This function is used to check whether the socket input is currently positioned at the out-of-band data mark.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

A bool indicating whether the socket is at the out-of-band data mark.

basic_raw_socket::available

Determine the number of bytes available for reading.

```
std::size_t available() const;
  » more...

std::size_t available(
    boost::system::error_code & ec) const;
  » more...
```

basic_raw_socket::available (1 of 2 overloads)

Inherited from basic_socket.

Determine the number of bytes available for reading.

```
std::size_t available() const;
```

This function is used to determine the number of bytes that may be read without blocking.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Return Value

The number of bytes that may be read without blocking, or 0 if an error occurs.

Exceptions

boost::system::system_error Thrown on failure.

basic_raw_socket::available (2 of 2 overloads)

Inherited from basic_socket.

Determine the number of bytes available for reading.

```
std::size_t available(
    boost::system::error_code & ec) const;
```

This function is used to determine the number of bytes that may be read without blocking.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes that may be read without blocking, or 0 if an error occurs.

basic_raw_socket::basic_raw_socket

Construct a `basic_raw_socket` without opening it.

```
explicit basic_raw_socket(
    boost::asio::io_service & io_service);
  » more...
```

Construct and open a `basic_raw_socket`.

```
basic_raw_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol);
  » more...
```

Construct a `basic_raw_socket`, opening it and binding it to the given local endpoint.

```
basic_raw_socket(
    boost::asio::io_service & io_service,
    const endpoint_type & endpoint);
  » more...
```

Construct a `basic_raw_socket` on an existing native socket.

```
basic_raw_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol,
    const native_handle_type & native_socket);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Move-construct a basic_raw_socket from another.

```
basic_raw_socket(
    basic_raw_socket && other);
  » more...
```

basic_raw_socket::basic_raw_socket (1 of 5 overloads)

Construct a basic_raw_socket without opening it.

```
basic_raw_socket(
    boost::asio::io_service & io_service);
```

This constructor creates a raw socket without opening it. The open() function must be called before data can be sent or received on the socket.

Parameters

io_service	The io_service object that the raw socket will use to dispatch handlers for any asynchronous operations performed on the socket.

basic_raw_socket::basic_raw_socket (2 of 5 overloads)

Construct and open a basic_raw_socket.

```
basic_raw_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol);
```

This constructor creates and opens a raw socket.

Parameters

io_service	The io_service object that the raw socket will use to dispatch handlers for any asynchronous operations performed on the socket.
protocol	An object specifying protocol parameters to be used.

Exceptions

boost::system::system_error	Thrown on failure.

basic_raw_socket::basic_raw_socket (3 of 5 overloads)

Construct a basic_raw_socket, opening it and binding it to the given local endpoint.

```
basic_raw_socket(
    boost::asio::io_service & io_service,
    const endpoint_type & endpoint);
```

This constructor creates a raw socket and automatically opens it bound to the specified endpoint on the local machine. The protocol used is the protocol associated with the given endpoint.

Parameters

io_service	The io_service object that the raw socket will use to dispatch handlers for any asynchronous operations performed on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

endpoint An endpoint on the local machine to which the raw socket will be bound.

Exceptions

boost::system::system_error Thrown on failure.

basic_raw_socket::basic_raw_socket (4 of 5 overloads)

Construct a `basic_raw_socket` on an existing native socket.

```
basic_raw_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol,
    const native_handle_type & native_socket);
```

This constructor creates a raw socket object to hold an existing native socket.

Parameters

io_service The `io_service` object that the raw socket will use to dispatch handlers for any asynchronous operations performed on the socket.

protocol An object specifying protocol parameters to be used.

native_socket The new underlying socket implementation.

Exceptions

boost::system::system_error Thrown on failure.

basic_raw_socket::basic_raw_socket (5 of 5 overloads)

Move-construct a `basic_raw_socket` from another.

```
basic_raw_socket(
    basic_raw_socket && other);
```

This constructor moves a raw socket from one object to another.

Parameters

other The other `basic_raw_socket` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_raw_socket(io_service&)` constructor.

basic_raw_socket::bind

Bind the socket to the given local endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void bind(
    const endpoint_type & endpoint);
  » more...

boost::system::error_code bind(
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::bind (1 of 2 overloads)

Inherited from basic_socket.

Bind the socket to the given local endpoint.

```
void bind(
    const endpoint_type & endpoint);
```

This function binds the socket to the specified endpoint on the local machine.

Parameters

endpoint An endpoint on the local machine to which the socket will be bound.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
socket.bind(boost::asio::ip::tcp::endpoint(
      boost::asio::ip::tcp::v4(), 12345));
```

basic_raw_socket::bind (2 of 2 overloads)

Inherited from basic_socket.

Bind the socket to the given local endpoint.

```
boost::system::error_code bind(
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
```

This function binds the socket to the specified endpoint on the local machine.

Parameters

endpoint An endpoint on the local machine to which the socket will be bound.

ec Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
boost::system::error_code ec;
socket.bind(boost::asio::ip::tcp::endpoint(
      boost::asio::ip::tcp::v4(), 12345), ec);
if (ec)
{
  // An error occurred.
}
```

basic_raw_socket::broadcast

Inherited from socket_base.

Socket option to permit sending of broadcast messages.

```
typedef implementation_defined broadcast;
```

Implements the SOL_SOCKET/SO_BROADCAST socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::bytes_readable

Inherited from socket_base.

IO control command to get the amount of data that can be read without blocking.

```
typedef implementation_defined bytes_readable;
```

Implements the FIONREAD IO control command.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::bytes_readable command(true);
socket.io_control(command);
std::size_t bytes_readable = command.get();
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::cancel

Cancel all asynchronous operations associated with the socket.

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::cancel (1 of 2 overloads)

Inherited from basic_socket.

Cancel all asynchronous operations associated with the socket.

```
void cancel();
```

This function causes all outstanding asynchronous connect, send and receive operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

Calls to `cancel()` will always fail with `boost::asio::error::operation_not_supported` when run on Windows XP, Windows Server 2003, and earlier versions of Windows, unless BOOST_ASIO_ENABLE_CANCELIO is defined. However, the CancelIo function has two issues that should be considered before enabling its use:

- It will only cancel asynchronous operations that were initiated in the current thread.
- It can appear to complete without error, but the request to cancel the unfinished operations may be silently ignored by the operating system. Whether it works or not seems to depend on the drivers that are installed.

For portable cancellation, consider using one of the following alternatives:

- Disable asio's I/O completion port backend by defining BOOST_ASIO_DISABLE_IOCP.
- Use the `close()` function to simultaneously cancel the outstanding operations and close the socket.

When running on Windows Vista, Windows Server 2008, and later, the CancelIoEx function is always used. This function does not have the problems described above.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket::cancel (2 of 2 overloads)

Inherited from basic_socket.

Cancel all asynchronous operations associated with the socket.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function causes all outstanding asynchronous connect, send and receive operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Parameters

ec Set to indicate what error occurred, if any.

Remarks

Calls to `cancel()` will always fail with `boost::asio::error::operation_not_supported` when run on Windows XP, Windows Server 2003, and earlier versions of Windows, unless BOOST_ASIO_ENABLE_CANCELIO is defined. However, the CancelIo function has two issues that should be considered before enabling its use:

- It will only cancel asynchronous operations that were initiated in the current thread.
- It can appear to complete without error, but the request to cancel the unfinished operations may be silently ignored by the operating system. Whether it works or not seems to depend on the drivers that are installed.

For portable cancellation, consider using one of the following alternatives:

- Disable asio's I/O completion port backend by defining BOOST_ASIO_DISABLE_IOCP.
- Use the `close()` function to simultaneously cancel the outstanding operations and close the socket.

When running on Windows Vista, Windows Server 2008, and later, the CancelIoEx function is always used. This function does not have the problems described above.

basic_raw_socket::close

Close the socket.

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::close (1 of 2 overloads)

Inherited from basic_socket.

Close the socket.

```
void close();
```

This function is used to close the socket. Any asynchronous send, receive or connect operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure. Note that, even if the function indicates an error, the underlying descriptor is closed.

Remarks

For portable behaviour with respect to graceful closure of a connected socket, call `shutdown()` before closing the socket.

basic_raw_socket::close (2 of 2 overloads)

Inherited from basic_socket.

Close the socket.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```

This function is used to close the socket. Any asynchronous send, receive or connect operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Parameters

ec Set to indicate what error occurred, if any. Note that, even if the function indicates an error, the underlying descriptor is closed.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
socket.close(ec);
if (ec)
{
  // An error occurred.
}
```

Remarks

For portable behaviour with respect to graceful closure of a connected socket, call `shutdown()` before closing the socket.

basic_raw_socket::connect

Connect the socket to the specified endpoint.

```
void connect(
    const endpoint_type & peer_endpoint);
  » more...

boost::system::error_code connect(
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::connect (1 of 2 overloads)

Inherited from basic_socket.

Connect the socket to the specified endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void connect(
    const endpoint_type & peer_endpoint);
```

This function is used to connect a socket to the specified remote endpoint. The function call will block until the connection is successfully made or an error occurs.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint	The remote endpoint to which the socket will be connected.

Exceptions

boost::system::system_error	Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.connect(endpoint);
```

basic_raw_socket::connect (2 of 2 overloads)

Inherited from basic_socket.

Connect the socket to the specified endpoint.

```
boost::system::error_code connect(
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
```

This function is used to connect a socket to the specified remote endpoint. The function call will block until the connection is successfully made or an error occurs.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint	The remote endpoint to which the socket will be connected.
ec	Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
boost::system::error_code ec;
socket.connect(endpoint, ec);
if (ec)
{
  // An error occurred.
}
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket::debug

Inherited from socket_base.

Socket option to enable socket-level debugging.

```
typedef implementation_defined debug;
```

Implements the SOL_SOCKET/SO_DEBUG socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::do_not_route

Inherited from socket_base.

Socket option to prevent routing, use local interfaces only.

```
typedef implementation_defined do_not_route;
```

Implements the SOL_SOCKET/SO_DONTROUTE socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option(true);
socket.set_option(option);
```

Getting the current option value:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::enable_connection_aborted

Inherited from socket_base.

Socket option to report aborted connections on accept.

```
typedef implementation_defined enable_connection_aborted;
```

Implements a custom socket option that determines whether or not an accept operation is permitted to fail with `boost::asio::error::connection_aborted`. By default the option is false.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option(true);
acceptor.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::endpoint_type

The endpoint type.

```
typedef Protocol::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

basic_raw_socket::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

basic_raw_socket::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

basic_raw_socket::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

basic_raw_socket::get_option

Get an option from the socket.

```
void get_option(
    GettableSocketOption & option) const;
  » more...

boost::system::error_code get_option(
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
  » more...
```

basic_raw_socket::get_option (1 of 2 overloads)

Inherited from basic_socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Get an option from the socket.

```
template<
    typename GettableSocketOption>
void get_option(
    GettableSocketOption & option) const;
```

This function is used to get the current value of an option on the socket.

Parameters

option The option value to be obtained from the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Getting the value of the SOL_SOCKET/SO_KEEPALIVE option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::keep_alive option;
socket.get_option(option);
bool is_set = option.get();
```

basic_raw_socket::get_option (2 of 2 overloads)

Inherited from basic_socket.

Get an option from the socket.

```
template<
    typename GettableSocketOption>
boost::system::error_code get_option(
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
```

This function is used to get the current value of an option on the socket.

Parameters

option The option value to be obtained from the socket.

ec Set to indicate what error occurred, if any.

Example

Getting the value of the SOL_SOCKET/SO_KEEPALIVE option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::keep_alive option;
boost::system::error_code ec;
socket.get_option(option, ec);
if (ec)
{
  // An error occurred.
}
bool is_set = option.get();
```

basic_raw_socket::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

basic_raw_socket::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

basic_raw_socket::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

basic_raw_socket::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

basic_raw_socket::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::io_control

Perform an IO control command on the socket.

```
void io_control(
    IoControlCommand & command);
  » more...

boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::io_control (1 of 2 overloads)

Inherited from basic_socket.

Perform an IO control command on the socket.

```
template<
    typename IoControlCommand>
void io_control(
    IoControlCommand & command);
```

This function is used to execute an IO control command on the socket.

Parameters

command The IO control command to be performed on the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Getting the number of bytes ready to read:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::bytes_readable command;
socket.io_control(command);
std::size_t bytes_readable = command.get();
```

basic_raw_socket::io_control (2 of 2 overloads)

Inherited from basic_socket.

Perform an IO control command on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
```

This function is used to execute an IO control command on the socket.

Parameters

command The IO control command to be performed on the socket.

ec Set to indicate what error occurred, if any.

Example

Getting the number of bytes ready to read:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::bytes_readable command;
boost::system::error_code ec;
socket.io_control(command, ec);
if (ec)
{
  // An error occurred.
}
std::size_t bytes_readable = command.get();
```

basic_raw_socket::is_open

Inherited from basic_socket.

Determine whether the socket is open.

```
bool is_open() const;
```

basic_raw_socket::keep_alive

Inherited from socket_base.

Socket option to send keep-alives.

```
typedef implementation_defined keep_alive;
```

Implements the SOL_SOCKET/SO_KEEPALIVE socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option(true);
socket.set_option(option);
```

Getting the current option value:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::linger

Inherited from socket_base.

Socket option to specify whether the socket lingers on close if unsent data is present.

```
typedef implementation_defined linger;
```

Implements the SOL_SOCKET/SO_LINGER socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option(true, 30);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option;
socket.get_option(option);
bool is_set = option.enabled();
unsigned short timeout = option.timeout();
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::local_endpoint

Get the local endpoint of the socket.

```
endpoint_type local_endpoint() const;
  » more...

endpoint_type local_endpoint(
    boost::system::error_code & ec) const;
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket::local_endpoint (1 of 2 overloads)

Inherited from basic_socket.

Get the local endpoint of the socket.

```
endpoint_type local_endpoint() const;
```

This function is used to obtain the locally bound endpoint of the socket.

Return Value

An object that represents the local endpoint of the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::endpoint endpoint = socket.local_endpoint();
```

basic_raw_socket::local_endpoint (2 of 2 overloads)

Inherited from basic_socket.

Get the local endpoint of the socket.

```
endpoint_type local_endpoint(
    boost::system::error_code & ec) const;
```

This function is used to obtain the locally bound endpoint of the socket.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

An object that represents the local endpoint of the socket. Returns a default-constructed endpoint object if an error occurred.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
boost::asio::ip::tcp::endpoint endpoint = socket.local_endpoint(ec);
if (ec)
{
  // An error occurred.
}
```

basic_raw_socket::lowest_layer

Get a reference to the lowest layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

basic_raw_socket::lowest_layer (1 of 2 overloads)

Inherited from basic_socket.

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

This function returns a reference to the lowest layer in a stack of layers. Since a `basic_socket` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

basic_raw_socket::lowest_layer (2 of 2 overloads)

Inherited from basic_socket.

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

This function returns a const reference to the lowest layer in a stack of layers. Since a `basic_socket` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A const reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

basic_raw_socket::lowest_layer_type

Inherited from basic_socket.

A `basic_socket` is always the lowest layer.

```
typedef basic_socket< Protocol, RawSocketService > lowest_layer_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_socket	Construct a basic_socket without opening it. Construct and open a basic_socket. Construct a basic_socket, opening it and binding it to the given local endpoint. Construct a basic_socket on an existing native socket. Move-construct a basic_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_socket from another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
remote_endpoint	Get the remote endpoint of the socket.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_socket	Protected destructor to prevent deletion through this type.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_socket` class template provides functionality that is common to both stream-oriented and datagram-oriented sockets.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket::max_connections

Inherited from socket_base.

The maximum length of the queue of pending incoming connections.

```
static const int max_connections = implementation_defined;
```

basic_raw_socket::message_do_not_route

Inherited from socket_base.

Specify that the data should not be subject to routing.

```
static const int message_do_not_route = implementation_defined;
```

basic_raw_socket::message_end_of_record

Inherited from socket_base.

Specifies that the data marks the end of a record.

```
static const int message_end_of_record = implementation_defined;
```

basic_raw_socket::message_flags

Inherited from socket_base.

Bitmask type for flags that can be passed to send and receive operations.

```
typedef int message_flags;
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::message_out_of_band

Inherited from socket_base.

Process out-of-band data.

```
static const int message_out_of_band = implementation_defined;
```

basic_raw_socket::message_peek

Inherited from socket_base.

Peek at incoming data without removing it from the input queue.

```
static const int message_peek = implementation_defined;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket::native

Inherited from basic_socket.

(Deprecated: Use `native_handle()`.) Get the native socket representation.

```
native_type native();
```

This function may be used to obtain the underlying representation of the socket. This is intended to allow access to native socket functionality that is not otherwise provided.

basic_raw_socket::native_handle

Inherited from basic_socket.

Get the native socket representation.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying representation of the socket. This is intended to allow access to native socket functionality that is not otherwise provided.

basic_raw_socket::native_handle_type

The native representation of a socket.

```
typedef RawSocketService::native_handle_type native_handle_type;
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::native_non_blocking

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking() const;
  » more...
```

Sets the non-blocking mode of the native socket implementation.

```
void native_non_blocking(
    bool mode);
  » more...

boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::native_non_blocking (1 of 3 overloads)

Inherited from basic_socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking() const;
```

This function is used to retrieve the non-blocking mode of the underlying native socket. This mode has no effect on the behaviour of the socket object's synchronous operations.

Return Value

`true` if the underlying socket is in non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Remarks

The current non-blocking mode is cached by the socket object. Consequently, the return value may be incorrect if the non-blocking mode was set directly on the native socket.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_raw_socket::native_non_blocking (2 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the native socket implementation.

```
void native_non_blocking(
    bool mode);
```

This function is used to modify the non-blocking mode of the underlying native socket. It has no effect on the behaviour of the socket object's synchronous operations.

Parameters

mode — If `true`, the underlying socket is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Exceptions

boost::system::system_error — Thrown on failure. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_raw_socket::native_non_blocking (3 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the native socket implementation.

```
boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

This function is used to modify the non-blocking mode of the underlying native socket. It has no effect on the behaviour of the socket object's synchronous operations.

Parameters

mode — If `true`, the underlying socket is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

ec — Set to indicate what error occurred, if any. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_raw_socket::native_type

(Deprecated: Use native_handle_type.) The native representation of a socket.

```
typedef RawSocketService::native_handle_type native_type;
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::non_blocking

Gets the non-blocking mode of the socket.

```
bool non_blocking() const;
  » more...
```

Sets the non-blocking mode of the socket.

```
void non_blocking(
    bool mode);
  » more...

boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::non_blocking (1 of 3 overloads)

Inherited from basic_socket.

Gets the non-blocking mode of the socket.

```
bool non_blocking() const;
```

Return Value

`true` if the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket::non_blocking (2 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the socket.

```
void non_blocking(
    bool mode);
```

Parameters

mode — If `true`, the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Exceptions

boost::system::system_error — Thrown on failure.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_raw_socket::non_blocking (3 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the socket.

```
boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

Parameters

mode — If `true`, the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

ec — Set to indicate what error occurred, if any.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_raw_socket::non_blocking_io

Inherited from socket_base.

(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.

```
typedef implementation_defined non_blocking_io;
```

Implements the FIONBIO IO control command.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::non_blocking_io command(true);
socket.io_control(command);
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::open

Open the socket using the specified protocol.

```
void open(
    const protocol_type & protocol = protocol_type());
  » more...

boost::system::error_code open(
    const protocol_type & protocol,
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::open (1 of 2 overloads)

Inherited from basic_socket.

Open the socket using the specified protocol.

```
void open(
    const protocol_type & protocol = protocol_type());
```

This function opens the socket so that it will use the specified protocol.

Parameters

protocol An object specifying protocol parameters to be used.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
```

basic_raw_socket::open (2 of 2 overloads)

Inherited from basic_socket.

Open the socket using the specified protocol.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code open(
    const protocol_type & protocol,
    boost::system::error_code & ec);
```

This function opens the socket so that it will use the specified protocol.

Parameters

protocol — An object specifying which protocol is to be used.

ec — Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::system::error_code ec;
socket.open(boost::asio::ip::tcp::v4(), ec);
if (ec)
{
  // An error occurred.
}
```

basic_raw_socket::operator=

Move-assign a `basic_raw_socket` from another.

```
basic_raw_socket & operator=(
    basic_raw_socket && other);
```

This assignment operator moves a raw socket from one object to another.

Parameters

other — The other `basic_raw_socket` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_raw_socket(io_service&)` constructor.

basic_raw_socket::protocol_type

The protocol type.

```
typedef Protocol protocol_type;
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::receive

Receive some data on a connected socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers);
  » more...

template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags);
  » more...

template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::receive (1 of 3 overloads)

Receive some data on a connected socket.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers);
```

This function is used to receive data on the raw socket. The function call will block until data has been received successfully or an error occurs.

Parameters

buffers One or more buffers into which the data will be received.

Return Value

The number of bytes received.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

The receive operation can only be used with a connected socket. Use the receive_from function to receive data on an unconnected raw socket.

Example

To receive into a single data buffer use the `buffer` function as follows:

```
socket.receive(boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket::receive (2 of 3 overloads)

Receive some data on a connected socket.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags);
```

This function is used to receive data on the raw socket. The function call will block until data has been received successfully or an error occurs.

Parameters

buffers One or more buffers into which the data will be received.

flags Flags specifying how the receive call is to be made.

Return Value

The number of bytes received.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

The receive operation can only be used with a connected socket. Use the receive_from function to receive data on an unconnected raw socket.

basic_raw_socket::receive (3 of 3 overloads)

Receive some data on a connected socket.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

This function is used to receive data on the raw socket. The function call will block until data has been received successfully or an error occurs.

Parameters

buffers One or more buffers into which the data will be received.

flags Flags specifying how the receive call is to be made.

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes received.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

The receive operation can only be used with a connected socket. Use the receive_from function to receive data on an unconnected raw socket.

basic_raw_socket::receive_buffer_size

Inherited from socket_base.

Socket option for the receive buffer size of a socket.

```
typedef implementation_defined receive_buffer_size;
```

Implements the SOL_SOCKET/SO_RCVBUF socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::receive_from

Receive raw data with the endpoint of the sender.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint);
  » more...

template<
    typename MutableBufferSequence>
std::size_t receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags);
  » more...

template<
    typename MutableBufferSequence>
std::size_t receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::receive_from (1 of 3 overloads)

Receive raw data with the endpoint of the sender.

```
template<
    typename MutableBufferSequence>
std::size_t receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint);
```

This function is used to receive raw data. The function call will block until data has been received successfully or an error occurs.

Parameters

buffers	One or more buffers into which the data will be received.
sender_endpoint	An endpoint object that receives the endpoint of the remote sender of the data.

Return Value

The number of bytes received.

Exceptions

boost::system::system_error	Thrown on failure.

Example

To receive into a single data buffer use the `buffer` function as follows:

```
boost::asio::ip::udp::endpoint sender_endpoint;
socket.receive_from(
    boost::asio::buffer(data, size), sender_endpoint);
```

See the `buffer` documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket::receive_from (2 of 3 overloads)

Receive raw data with the endpoint of the sender.

```
template<
    typename MutableBufferSequence>
std::size_t receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags);
```

This function is used to receive raw data. The function call will block until data has been received successfully or an error occurs.

Parameters

buffers	One or more buffers into which the data will be received.
sender_endpoint	An endpoint object that receives the endpoint of the remote sender of the data.
flags	Flags specifying how the receive call is to be made.

Return Value

The number of bytes received.

Exceptions

boost::system::system_error	Thrown on failure.

basic_raw_socket::receive_from (3 of 3 overloads)

Receive raw data with the endpoint of the sender.

```
template<
    typename MutableBufferSequence>
std::size_t receive_from(
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

This function is used to receive raw data. The function call will block until data has been received successfully or an error occurs.

Parameters

buffers	One or more buffers into which the data will be received.
sender_endpoint	An endpoint object that receives the endpoint of the remote sender of the data.
flags	Flags specifying how the receive call is to be made.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes received.

basic_raw_socket::receive_low_watermark

Inherited from socket_base.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Socket option for the receive low watermark.

```
typedef implementation_defined receive_low_watermark;
```

Implements the SOL_SOCKET/SO_RCVLOWAT socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::remote_endpoint

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint() const;
  » more...

endpoint_type remote_endpoint(
    boost::system::error_code & ec) const;
  » more...
```

basic_raw_socket::remote_endpoint (1 of 2 overloads)

Inherited from basic_socket.

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint() const;
```

This function is used to obtain the remote endpoint of the socket.

Return Value

An object that represents the remote endpoint of the socket.

Exceptions

boost::system::system_error	Thrown on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::endpoint endpoint = socket.remote_endpoint();
```

basic_raw_socket::remote_endpoint (2 of 2 overloads)

Inherited from basic_socket.

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint(
    boost::system::error_code & ec) const;
```

This function is used to obtain the remote endpoint of the socket.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

An object that represents the remote endpoint of the socket. Returns a default-constructed endpoint object if an error occurred.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
boost::asio::ip::tcp::endpoint endpoint = socket.remote_endpoint(ec);
if (ec)
{
  // An error occurred.
}
```

basic_raw_socket::reuse_address

Inherited from socket_base.

Socket option to allow the socket to be bound to an address that is already in use.

```
typedef implementation_defined reuse_address;
```

Implements the SOL_SOCKET/SO_REUSEADDR socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option(true);
acceptor.set_option(option);
```

Getting the current option value:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::send

Send some data on a connected socket.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers);
  » more...

template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags);
  » more...

template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::send (1 of 3 overloads)

Send some data on a connected socket.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers);
```

This function is used to send data on the raw socket. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers One ore more data buffers to be sent on the socket.

Return Value

The number of bytes sent.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

The send operation can only be used with a connected socket. Use the send_to function to send data on an unconnected raw socket.

Example

To send a single data buffer use the `buffer` function as follows:

```
socket.send(boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on sending multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_raw_socket::send (2 of 3 overloads)

Send some data on a connected socket.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags);
```

This function is used to send data on the raw socket. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers	One ore more data buffers to be sent on the socket.
flags	Flags specifying how the send call is to be made.

Return Value

The number of bytes sent.

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

The send operation can only be used with a connected socket. Use the send_to function to send data on an unconnected raw socket.

basic_raw_socket::send (3 of 3 overloads)

Send some data on a connected socket.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This function is used to send data on the raw socket. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers	One or more data buffers to be sent on the socket.
flags	Flags specifying how the send call is to be made.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes sent.

Remarks

The send operation can only be used with a connected socket. Use the send_to function to send data on an unconnected raw socket.

basic_raw_socket::send_buffer_size

Inherited from socket_base.

Socket option for the send buffer size of a socket.

```
typedef implementation_defined send_buffer_size;
```

Implements the SOL_SOCKET/SO_SNDBUF socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::send_low_watermark

Inherited from socket_base.

Socket option for the send low watermark.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef implementation_defined send_low_watermark;
```

Implements the SOL_SOCKET/SO_SNDLOWAT socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_raw_socket::send_to

Send raw data to the specified endpoint.

```
template<
    typename ConstBufferSequence>
std::size_t send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination);
  » more...

template<
    typename ConstBufferSequence>
std::size_t send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags);
  » more...

template<
    typename ConstBufferSequence>
std::size_t send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::send_to (1 of 3 overloads)

Send raw data to the specified endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination);
```

This function is used to send raw data to the specified remote endpoint. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers	One or more data buffers to be sent to the remote endpoint.
destination	The remote endpoint to which the data will be sent.

Return Value

The number of bytes sent.

Exceptions

boost::system::system_error	Thrown on failure.

Example

To send a single data buffer use the `buffer` function as follows:

```
boost::asio::ip::udp::endpoint destination(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.send_to(boost::asio::buffer(data, size), destination);
```

See the `buffer` documentation for information on sending multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_raw_socket::send_to (2 of 3 overloads)

Send raw data to the specified endpoint.

```
template<
    typename ConstBufferSequence>
std::size_t send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags);
```

This function is used to send raw data to the specified remote endpoint. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers	One or more data buffers to be sent to the remote endpoint.
destination	The remote endpoint to which the data will be sent.
flags	Flags specifying how the send call is to be made.

Return Value

The number of bytes sent.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

basic_raw_socket::send_to (3 of 3 overloads)

Send raw data to the specified endpoint.

```
template<
    typename ConstBufferSequence>
std::size_t send_to(
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

This function is used to send raw data to the specified remote endpoint. The function call will block until the data has been sent successfully or an error occurs.

Parameters

buffers	One or more data buffers to be sent to the remote endpoint.
destination	The remote endpoint to which the data will be sent.
flags	Flags specifying how the send call is to be made.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes sent.

basic_raw_socket::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

basic_raw_socket::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef RawSocketService service_type;
```

Requirements

Header: `boost/asio/basic_raw_socket.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket::set_option

Set an option on the socket.

```
void set_option(
    const SettableSocketOption & option);
  » more...

boost::system::error_code set_option(
    const SettableSocketOption & option,
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::set_option (1 of 2 overloads)

Inherited from basic_socket.

Set an option on the socket.

```
template<
    typename SettableSocketOption>
void set_option(
    const SettableSocketOption & option);
```

This function is used to set an option on the socket.

Parameters

option The new option value to be set on the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Setting the IPPROTO_TCP/TCP_NODELAY option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::no_delay option(true);
socket.set_option(option);
```

basic_raw_socket::set_option (2 of 2 overloads)

Inherited from basic_socket.

Set an option on the socket.

```
template<
    typename SettableSocketOption>
boost::system::error_code set_option(
    const SettableSocketOption & option,
    boost::system::error_code & ec);
```

This function is used to set an option on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

option The new option value to be set on the socket.

ec Set to indicate what error occurred, if any.

Example

Setting the IPPROTO_TCP/TCP_NODELAY option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::no_delay option(true);
boost::system::error_code ec;
socket.set_option(option, ec);
if (ec)
{
  // An error occurred.
}
```

basic_raw_socket::shutdown

Disable sends or receives on the socket.

```
void shutdown(
    shutdown_type what);
  » more...

boost::system::error_code shutdown(
    shutdown_type what,
    boost::system::error_code & ec);
  » more...
```

basic_raw_socket::shutdown (1 of 2 overloads)

Inherited from basic_socket.

Disable sends or receives on the socket.

```
void shutdown(
    shutdown_type what);
```

This function is used to disable send operations, receive operations, or both.

Parameters

what Determines what types of operation will no longer be allowed.

Exceptions

boost::system::system_error Thrown on failure.

Example

Shutting down the send side of the socket:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
socket.shutdown(boost::asio::ip::tcp::socket::shutdown_send);
```

basic_raw_socket::shutdown (2 of 2 overloads)

Inherited from basic_socket.

Disable sends or receives on the socket.

```
boost::system::error_code shutdown(
    shutdown_type what,
    boost::system::error_code & ec);
```

This function is used to disable send operations, receive operations, or both.

Parameters

what Determines what types of operation will no longer be allowed.

ec Set to indicate what error occurred, if any.

Example

Shutting down the send side of the socket:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
socket.shutdown(boost::asio::ip::tcp::socket::shutdown_send, ec);
if (ec)
{
  // An error occurred.
}
```

basic_raw_socket::shutdown_type

Inherited from socket_base.

Different ways a socket may be shutdown.

```
enum shutdown_type
```

Values

shutdown_receive Shutdown the receive side of the socket.

shutdown_send Shutdown the send side of the socket.

shutdown_both Shutdown both send and receive on the socket.

basic_seq_packet_socket

Provides sequenced packet socket functionality.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename Protocol,
    typename SeqPacketSocketService = seq_packet_socket_service<Protocol>>
class basic_seq_packet_socket :
  public basic_socket< Protocol, SeqPacketSocketService >
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
async_receive	Start an asynchronous receive.
async_send	Start an asynchronous send.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_seq_packet_socket	Construct a basic_seq_packet_socket without opening it. Construct and open a basic_seq_packet_socket. Construct a basic_seq_packet_socket, opening it and binding it to the given local endpoint. Construct a basic_seq_packet_socket on an existing native socket. Move-construct a basic_seq_packet_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_seq_packet_socket from another.
receive	Receive some data on the socket. Receive some data on a connected socket.
remote_endpoint	Get the remote endpoint of the socket.
send	Send some data on the socket.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The basic_seq_packet_socket class template provides asynchronous and blocking sequenced packet socket functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: boost/asio/basic_seq_packet_socket.hpp

Convenience header: boost/asio.hpp

basic_seq_packet_socket::assign

Assign an existing native socket to the socket.

```
void assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket);
  » more...

boost::system::error_code assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
  » more...
```

basic_seq_packet_socket::assign (1 of 2 overloads)

Inherited from basic_socket.

Assign an existing native socket to the socket.

```
void assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket);
```

basic_seq_packet_socket::assign (2 of 2 overloads)

Inherited from basic_socket.

Assign an existing native socket to the socket.

```
boost::system::error_code assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
```

basic_seq_packet_socket::async_connect

Inherited from basic_socket.

Start an asynchronous connect.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConnectHandler>
void async_connect(
    const endpoint_type & peer_endpoint,
    ConnectHandler handler);
```

This function is used to asynchronously connect a socket to the specified remote endpoint. The function call always returns immediately.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint — The remote endpoint to which the socket will be connected. Copies will be made of the endpoint object as required.

handler — The handler to be called when the connection operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error // Result of operation
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

```
void connect_handler(const boost::system::error_code& error)
{
  if (!error)
  {
    // Connect succeeded.
  }
}

...

boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.async_connect(endpoint, connect_handler);
```

basic_seq_packet_socket::async_receive

Start an asynchronous receive.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags & out_flags,
    ReadHandler handler);
  » more...

template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags in_flags,
    socket_base::message_flags & out_flags,
    ReadHandler handler);
  » more...
```

basic_seq_packet_socket::async_receive (1 of 2 overloads)

Start an asynchronous receive.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags & out_flags,
    ReadHandler handler);
```

This function is used to asynchronously receive data from the sequenced packet socket. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be received. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

out_flags — Once the asynchronous operation completes, contains flags associated with the received data. For example, if the `socket_base::message_end_of_record` bit is set then the received data marks the end of a record. The caller must guarantee that the referenced variable remains valid until the handler is called.

handler — The handler to be called when the receive operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes received.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To receive into a single data buffer use the `buffer` function as follows:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
socket.async_receive(boost::asio::buffer(data, size), out_flags, handler);
```

See the buffer documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_seq_packet_socket::async_receive (2 of 2 overloads)

Start an asynchronous receive.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags in_flags,
    socket_base::message_flags & out_flags,
    ReadHandler handler);
```

This function is used to asynchronously receive data from the sequenced data socket. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be received. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

in_flags — Flags specifying how the receive call is to be made.

out_flags — Once the asynchronous operation completes, contains flags associated with the received data. For example, if the `socket_base::message_end_of_record` bit is set then the received data marks the end of a record. The caller must guarantee that the referenced variable remains valid until the handler is called.

handler — The handler to be called when the receive operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes received.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To receive into a single data buffer use the `buffer` function as follows:

```
socket.async_receive(
    boost::asio::buffer(data, size),
    0, out_flags, handler);
```

See the buffer documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_seq_packet_socket::async_send

Start an asynchronous send.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    WriteHandler handler);
```

This function is used to asynchronously send data on the sequenced packet socket. The function call always returns immediately.

Parameters

buffers — One or more data buffers to be sent on the socket. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

flags — Flags specifying how the send call is to be made.

handler — The handler to be called when the send operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes sent.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

To send a single data buffer use the `buffer` function as follows:

```
socket.async_send(boost::asio::buffer(data, size), 0, handler);
```

See the `buffer` documentation for information on sending multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_seq_packet_socket::at_mark

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark() const;
  » more...

bool at_mark(
    boost::system::error_code & ec) const;
  » more...
```

basic_seq_packet_socket::at_mark (1 of 2 overloads)

Inherited from basic_socket.

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark() const;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This function is used to check whether the socket input is currently positioned at the out-of-band data mark.

Return Value

A bool indicating whether the socket is at the out-of-band data mark.

Exceptions

boost::system::system_error Thrown on failure.

basic_seq_packet_socket::at_mark (2 of 2 overloads)

Inherited from basic_socket.

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark(
    boost::system::error_code & ec) const;
```

This function is used to check whether the socket input is currently positioned at the out-of-band data mark.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

A bool indicating whether the socket is at the out-of-band data mark.

basic_seq_packet_socket::available

Determine the number of bytes available for reading.

```
std::size_t available() const;
  » more...

std::size_t available(
    boost::system::error_code & ec) const;
  » more...
```

basic_seq_packet_socket::available (1 of 2 overloads)

Inherited from basic_socket.

Determine the number of bytes available for reading.

```
std::size_t available() const;
```

This function is used to determine the number of bytes that may be read without blocking.

Return Value

The number of bytes that may be read without blocking, or 0 if an error occurs.

Exceptions

boost::system::system_error Thrown on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_seq_packet_socket::available (2 of 2 overloads)

Inherited from basic_socket.

Determine the number of bytes available for reading.

```
std::size_t available(
    boost::system::error_code & ec) const;
```

This function is used to determine the number of bytes that may be read without blocking.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes that may be read without blocking, or 0 if an error occurs.

basic_seq_packet_socket::basic_seq_packet_socket

Construct a `basic_seq_packet_socket` without opening it.

```
explicit basic_seq_packet_socket(
    boost::asio::io_service & io_service);
  » more...
```

Construct and open a `basic_seq_packet_socket`.

```
basic_seq_packet_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol);
  » more...
```

Construct a `basic_seq_packet_socket`, opening it and binding it to the given local endpoint.

```
basic_seq_packet_socket(
    boost::asio::io_service & io_service,
    const endpoint_type & endpoint);
  » more...
```

Construct a `basic_seq_packet_socket` on an existing native socket.

```
basic_seq_packet_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol,
    const native_handle_type & native_socket);
  » more...
```

Move-construct a `basic_seq_packet_socket` from another.

```
basic_seq_packet_socket(
    basic_seq_packet_socket && other);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_seq_packet_socket::basic_seq_packet_socket (1 of 5 overloads)

Construct a basic_seq_packet_socket without opening it.

```
basic_seq_packet_socket(
    boost::asio::io_service & io_service);
```

This constructor creates a sequenced packet socket without opening it. The socket needs to be opened and then connected or accepted before data can be sent or received on it.

Parameters

io_service — The io_service object that the sequenced packet socket will use to dispatch handlers for any asynchronous operations performed on the socket.

basic_seq_packet_socket::basic_seq_packet_socket (2 of 5 overloads)

Construct and open a basic_seq_packet_socket.

```
basic_seq_packet_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol);
```

This constructor creates and opens a sequenced_packet socket. The socket needs to be connected or accepted before data can be sent or received on it.

Parameters

io_service — The io_service object that the sequenced packet socket will use to dispatch handlers for any asynchronous operations performed on the socket.

protocol — An object specifying protocol parameters to be used.

Exceptions

boost::system::system_error — Thrown on failure.

basic_seq_packet_socket::basic_seq_packet_socket (3 of 5 overloads)

Construct a basic_seq_packet_socket, opening it and binding it to the given local endpoint.

```
basic_seq_packet_socket(
    boost::asio::io_service & io_service,
    const endpoint_type & endpoint);
```

This constructor creates a sequenced packet socket and automatically opens it bound to the specified endpoint on the local machine. The protocol used is the protocol associated with the given endpoint.

Parameters

io_service — The io_service object that the sequenced packet socket will use to dispatch handlers for any asynchronous operations performed on the socket.

endpoint — An endpoint on the local machine to which the sequenced packet socket will be bound.

Exceptions

boost::system::system_error — Thrown on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_seq_packet_socket::basic_seq_packet_socket (4 of 5 overloads)

Construct a `basic_seq_packet_socket` on an existing native socket.

```
basic_seq_packet_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol,
    const native_handle_type & native_socket);
```

This constructor creates a sequenced packet socket object to hold an existing native socket.

Parameters

io_service	The `io_service` object that the sequenced packet socket will use to dispatch handlers for any asynchronous operations performed on the socket.
protocol	An object specifying protocol parameters to be used.
native_socket	The new underlying socket implementation.

Exceptions

boost::system::system_error	Thrown on failure.

basic_seq_packet_socket::basic_seq_packet_socket (5 of 5 overloads)

Move-construct a `basic_seq_packet_socket` from another.

```
basic_seq_packet_socket(
    basic_seq_packet_socket && other);
```

This constructor moves a sequenced packet socket from one object to another.

Parameters

other	The other `basic_seq_packet_socket` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_seq_packet_socket(io_service&) constructor`.

basic_seq_packet_socket::bind

Bind the socket to the given local endpoint.

```
void bind(
    const endpoint_type & endpoint);
  » more...

boost::system::error_code bind(
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
  » more...
```

basic_seq_packet_socket::bind (1 of 2 overloads)

Inherited from basic_socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Bind the socket to the given local endpoint.

```
void bind(
    const endpoint_type & endpoint);
```

This function binds the socket to the specified endpoint on the local machine.

Parameters

endpoint An endpoint on the local machine to which the socket will be bound.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
socket.bind(boost::asio::ip::tcp::endpoint(
      boost::asio::ip::tcp::v4(), 12345));
```

basic_seq_packet_socket::bind (2 of 2 overloads)

Inherited from basic_socket.

Bind the socket to the given local endpoint.

```
boost::system::error_code bind(
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
```

This function binds the socket to the specified endpoint on the local machine.

Parameters

endpoint An endpoint on the local machine to which the socket will be bound.

ec Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
boost::system::error_code ec;
socket.bind(boost::asio::ip::tcp::endpoint(
      boost::asio::ip::tcp::v4(), 12345), ec);
if (ec)
{
  // An error occurred.
}
```

basic_seq_packet_socket::broadcast

Inherited from socket_base.

Socket option to permit sending of broadcast messages.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef implementation_defined broadcast;
```

Implements the SOL_SOCKET/SO_BROADCAST socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::bytes_readable

Inherited from socket_base.

IO control command to get the amount of data that can be read without blocking.

```
typedef implementation_defined bytes_readable;
```

Implements the FIONREAD IO control command.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::bytes_readable command(true);
socket.io_control(command);
std::size_t bytes_readable = command.get();
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::cancel

Cancel all asynchronous operations associated with the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

basic_seq_packet_socket::cancel (1 of 2 overloads)

Inherited from basic_socket.

Cancel all asynchronous operations associated with the socket.

```
void cancel();
```

This function causes all outstanding asynchronous connect, send and receive operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

Calls to `cancel()` will always fail with `boost::asio::error::operation_not_supported` when run on Windows XP, Windows Server 2003, and earlier versions of Windows, unless BOOST_ASIO_ENABLE_CANCELIO is defined. However, the CancelIo function has two issues that should be considered before enabling its use:

- It will only cancel asynchronous operations that were initiated in the current thread.
- It can appear to complete without error, but the request to cancel the unfinished operations may be silently ignored by the operating system. Whether it works or not seems to depend on the drivers that are installed.

For portable cancellation, consider using one of the following alternatives:

- Disable asio's I/O completion port backend by defining BOOST_ASIO_DISABLE_IOCP.
- Use the `close()` function to simultaneously cancel the outstanding operations and close the socket.

When running on Windows Vista, Windows Server 2008, and later, the CancelIoEx function is always used. This function does not have the problems described above.

basic_seq_packet_socket::cancel (2 of 2 overloads)

Inherited from basic_socket.

Cancel all asynchronous operations associated with the socket.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function causes all outstanding asynchronous connect, send and receive operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Parameters

ec	Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

Calls to `cancel()` will always fail with `boost::asio::error::operation_not_supported` when run on Windows XP, Windows Server 2003, and earlier versions of Windows, unless BOOST_ASIO_ENABLE_CANCELIO is defined. However, the CancelIo function has two issues that should be considered before enabling its use:

- It will only cancel asynchronous operations that were initiated in the current thread.
- It can appear to complete without error, but the request to cancel the unfinished operations may be silently ignored by the operating system. Whether it works or not seems to depend on the drivers that are installed.

For portable cancellation, consider using one of the following alternatives:

- Disable asio's I/O completion port backend by defining BOOST_ASIO_DISABLE_IOCP.
- Use the `close()` function to simultaneously cancel the outstanding operations and close the socket.

When running on Windows Vista, Windows Server 2008, and later, the CancelIoEx function is always used. This function does not have the problems described above.

basic_seq_packet_socket::close

Close the socket.

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

basic_seq_packet_socket::close (1 of 2 overloads)

Inherited from basic_socket.

Close the socket.

```
void close();
```

This function is used to close the socket. Any asynchronous send, receive or connect operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure. Note that, even if the function indicates an error, the underlying descriptor is closed.

Remarks

For portable behaviour with respect to graceful closure of a connected socket, call `shutdown()` before closing the socket.

basic_seq_packet_socket::close (2 of 2 overloads)

Inherited from basic_socket.

Close the socket.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This function is used to close the socket. Any asynchronous send, receive or connect operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Parameters

ec Set to indicate what error occurred, if any. Note that, even if the function indicates an error, the underlying descriptor is closed.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
socket.close(ec);
if (ec)
{
  // An error occurred.
}
```

Remarks

For portable behaviour with respect to graceful closure of a connected socket, call `shutdown()` before closing the socket.

basic_seq_packet_socket::connect

Connect the socket to the specified endpoint.

```
void connect(
    const endpoint_type & peer_endpoint);
  » more...

boost::system::error_code connect(
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
  » more...
```

basic_seq_packet_socket::connect (1 of 2 overloads)

Inherited from basic_socket.

Connect the socket to the specified endpoint.

```
void connect(
    const endpoint_type & peer_endpoint);
```

This function is used to connect a socket to the specified remote endpoint. The function call will block until the connection is successfully made or an error occurs.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint The remote endpoint to which the socket will be connected.

Exceptions

boost::system::system_error Thrown on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.connect(endpoint);
```

basic_seq_packet_socket::connect (2 of 2 overloads)

Inherited from basic_socket.

Connect the socket to the specified endpoint.

```
boost::system::error_code connect(
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
```

This function is used to connect a socket to the specified remote endpoint. The function call will block until the connection is successfully made or an error occurs.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint	The remote endpoint to which the socket will be connected.
ec	Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
boost::system::error_code ec;
socket.connect(endpoint, ec);
if (ec)
{
  // An error occurred.
}
```

basic_seq_packet_socket::debug

Inherited from socket_base.

Socket option to enable socket-level debugging.

```
typedef implementation_defined debug;
```

Implements the SOL_SOCKET/SO_DEBUG socket option.

Examples

Setting the option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::do_not_route

Inherited from socket_base.

Socket option to prevent routing, use local interfaces only.

```
typedef implementation_defined do_not_route;
```

Implements the SOL_SOCKET/SO_DONTROUTE socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::enable_connection_aborted

Inherited from socket_base.

Socket option to report aborted connections on accept.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef implementation_defined enable_connection_aborted;
```

Implements a custom socket option that determines whether or not an accept operation is permitted to fail with `boost::asio::error::connection_aborted`. By default the option is false.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option(true);
acceptor.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::endpoint_type

The endpoint type.

```
typedef Protocol::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

basic_seq_packet_socket::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
implementation_type & get_implementation();
```

basic_seq_packet_socket::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

basic_seq_packet_socket::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

basic_seq_packet_socket::get_option

Get an option from the socket.

```
void get_option(
    GettableSocketOption & option) const;
  » more...

boost::system::error_code get_option(
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
  » more...
```

basic_seq_packet_socket::get_option (1 of 2 overloads)

Inherited from basic_socket.

Get an option from the socket.

```
template<
    typename GettableSocketOption>
void get_option(
    GettableSocketOption & option) const;
```

This function is used to get the current value of an option on the socket.

Parameters

option The option value to be obtained from the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Example

Getting the value of the SOL_SOCKET/SO_KEEPALIVE option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::keep_alive option;
socket.get_option(option);
bool is_set = option.get();
```

basic_seq_packet_socket::get_option (2 of 2 overloads)

Inherited from basic_socket.

Get an option from the socket.

```
template<
    typename GettableSocketOption>
boost::system::error_code get_option(
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
```

This function is used to get the current value of an option on the socket.

Parameters

option	The option value to be obtained from the socket.
ec	Set to indicate what error occurred, if any.

Example

Getting the value of the SOL_SOCKET/SO_KEEPALIVE option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::keep_alive option;
boost::system::error_code ec;
socket.get_option(option, ec);
if (ec)
{
  // An error occurred.
}
bool is_set = option.get();
```

basic_seq_packet_socket::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_seq_packet_socket::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

basic_seq_packet_socket::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

basic_seq_packet_socket::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

basic_seq_packet_socket::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::io_control

Perform an IO control command on the socket.

```
void io_control(
    IoControlCommand & command);
  » more...

boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
  » more...
```

basic_seq_packet_socket::io_control (1 of 2 overloads)

Inherited from basic_socket.

Perform an IO control command on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename IoControlCommand>
void io_control(
    IoControlCommand & command);
```

This function is used to execute an IO control command on the socket.

Parameters

command The IO control command to be performed on the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Getting the number of bytes ready to read:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::bytes_readable command;
socket.io_control(command);
std::size_t bytes_readable = command.get();
```

basic_seq_packet_socket::io_control (2 of 2 overloads)

Inherited from basic_socket.

Perform an IO control command on the socket.

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
```

This function is used to execute an IO control command on the socket.

Parameters

command The IO control command to be performed on the socket.

ec Set to indicate what error occurred, if any.

Example

Getting the number of bytes ready to read:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::bytes_readable command;
boost::system::error_code ec;
socket.io_control(command, ec);
if (ec)
{
  // An error occurred.
}
std::size_t bytes_readable = command.get();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_seq_packet_socket::is_open

Inherited from basic_socket.

Determine whether the socket is open.

```
bool is_open() const;
```

basic_seq_packet_socket::keep_alive

Inherited from socket_base.

Socket option to send keep-alives.

```
typedef implementation_defined keep_alive;
```

Implements the SOL_SOCKET/SO_KEEPALIVE socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::linger

Inherited from socket_base.

Socket option to specify whether the socket lingers on close if unsent data is present.

```
typedef implementation_defined linger;
```

Implements the SOL_SOCKET/SO_LINGER socket option.

Examples

Setting the option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option(true, 30);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option;
socket.get_option(option);
bool is_set = option.enabled();
unsigned short timeout = option.timeout();
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::local_endpoint

Get the local endpoint of the socket.

```
endpoint_type local_endpoint() const;
  » more...

endpoint_type local_endpoint(
    boost::system::error_code & ec) const;
  » more...
```

basic_seq_packet_socket::local_endpoint (1 of 2 overloads)

Inherited from basic_socket.

Get the local endpoint of the socket.

```
endpoint_type local_endpoint() const;
```

This function is used to obtain the locally bound endpoint of the socket.

Return Value

An object that represents the local endpoint of the socket.

Exceptions

boost::system::system_error	Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::endpoint endpoint = socket.local_endpoint();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_seq_packet_socket::local_endpoint (2 of 2 overloads)

Inherited from basic_socket.

Get the local endpoint of the socket.

```
endpoint_type local_endpoint(
    boost::system::error_code & ec) const;
```

This function is used to obtain the locally bound endpoint of the socket.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

An object that represents the local endpoint of the socket. Returns a default-constructed endpoint object if an error occurred.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
boost::asio::ip::tcp::endpoint endpoint = socket.local_endpoint(ec);
if (ec)
{
  // An error occurred.
}
```

basic_seq_packet_socket::lowest_layer

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

basic_seq_packet_socket::lowest_layer (1 of 2 overloads)

Inherited from basic_socket.

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

This function returns a reference to the lowest layer in a stack of layers. Since a `basic_socket` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_seq_packet_socket::lowest_layer (2 of 2 overloads)

Inherited from basic_socket.

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

This function returns a const reference to the lowest layer in a stack of layers. Since a `basic_socket` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A const reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

basic_seq_packet_socket::lowest_layer_type

Inherited from basic_socket.

A `basic_socket` is always the lowest layer.

```
typedef basic_socket< Protocol, SeqPacketSocketService > lowest_layer_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_socket	Construct a basic_socket without opening it. Construct and open a basic_socket. Construct a basic_socket, opening it and binding it to the given local endpoint. Construct a basic_socket on an existing native socket. Move-construct a basic_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_socket from another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
remote_endpoint	Get the remote endpoint of the socket.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_socket	Protected destructor to prevent deletion through this type.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_socket` class template provides functionality that is common to both stream-oriented and datagram-oriented sockets.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_seq_packet_socket::max_connections

Inherited from socket_base.

The maximum length of the queue of pending incoming connections.

```
static const int max_connections = implementation_defined;
```

basic_seq_packet_socket::message_do_not_route

Inherited from socket_base.

Specify that the data should not be subject to routing.

```
static const int message_do_not_route = implementation_defined;
```

basic_seq_packet_socket::message_end_of_record

Inherited from socket_base.

Specifies that the data marks the end of a record.

```
static const int message_end_of_record = implementation_defined;
```

basic_seq_packet_socket::message_flags

Inherited from socket_base.

Bitmask type for flags that can be passed to send and receive operations.

```
typedef int message_flags;
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::message_out_of_band

Inherited from socket_base.

Process out-of-band data.

```
static const int message_out_of_band = implementation_defined;
```

basic_seq_packet_socket::message_peek

Inherited from socket_base.

Peek at incoming data without removing it from the input queue.

```
static const int message_peek = implementation_defined;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_seq_packet_socket::native

Inherited from basic_socket.

(Deprecated: Use `native_handle()`.) Get the native socket representation.

```
native_type native();
```

This function may be used to obtain the underlying representation of the socket. This is intended to allow access to native socket functionality that is not otherwise provided.

basic_seq_packet_socket::native_handle

Inherited from basic_socket.

Get the native socket representation.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying representation of the socket. This is intended to allow access to native socket functionality that is not otherwise provided.

basic_seq_packet_socket::native_handle_type

The native representation of a socket.

```
typedef SeqPacketSocketService::native_handle_type native_handle_type;
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::native_non_blocking

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking() const;
  » more...
```

Sets the non-blocking mode of the native socket implementation.

```
void native_non_blocking(
    bool mode);
  » more...

boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

basic_seq_packet_socket::native_non_blocking (1 of 3 overloads)

Inherited from basic_socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking() const;
```

This function is used to retrieve the non-blocking mode of the underlying native socket. This mode has no effect on the behaviour of the socket object's synchronous operations.

Return Value

`true` if the underlying socket is in non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Remarks

The current non-blocking mode is cached by the socket object. Consequently, the return value may be incorrect if the non-blocking mode was set directly on the native socket.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_seq_packet_socket::native_non_blocking (2 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the native socket implementation.

```
void native_non_blocking(
    bool mode);
```

This function is used to modify the non-blocking mode of the underlying native socket. It has no effect on the behaviour of the socket object's synchronous operations.

Parameters

mode — If `true`, the underlying socket is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Exceptions

boost::system::system_error — Thrown on failure. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_seq_packet_socket::native_non_blocking (3 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the native socket implementation.

```
boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

This function is used to modify the non-blocking mode of the underlying native socket. It has no effect on the behaviour of the socket object's synchronous operations.

Parameters

mode — If `true`, the underlying socket is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

ec — Set to indicate what error occurred, if any. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_seq_packet_socket::native_type

(Deprecated: Use native_handle_type.) The native representation of a socket.

```
typedef SeqPacketSocketService::native_handle_type native_type;
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::non_blocking

Gets the non-blocking mode of the socket.

```
bool non_blocking() const;
  » more...
```

Sets the non-blocking mode of the socket.

```
void non_blocking(
    bool mode);
  » more...

boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

basic_seq_packet_socket::non_blocking (1 of 3 overloads)

Inherited from basic_socket.

Gets the non-blocking mode of the socket.

```
bool non_blocking() const;
```

Return Value

`true` if the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_seq_packet_socket::non_blocking (2 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the socket.

```
void non_blocking(
    bool mode);
```

Parameters

mode — If `true`, the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Exceptions

boost::system::system_error — Thrown on failure.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_seq_packet_socket::non_blocking (3 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the socket.

```
boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

Parameters

mode — If `true`, the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

ec — Set to indicate what error occurred, if any.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_seq_packet_socket::non_blocking_io

Inherited from socket_base.

(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.

```
typedef implementation_defined non_blocking_io;
```

Implements the FIONBIO IO control command.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::non_blocking_io command(true);
socket.io_control(command);
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::open

Open the socket using the specified protocol.

```
void open(
    const protocol_type & protocol = protocol_type());
  » more...

boost::system::error_code open(
    const protocol_type & protocol,
    boost::system::error_code & ec);
  » more...
```

basic_seq_packet_socket::open (1 of 2 overloads)

Inherited from basic_socket.

Open the socket using the specified protocol.

```
void open(
    const protocol_type & protocol = protocol_type());
```

This function opens the socket so that it will use the specified protocol.

Parameters

protocol An object specifying protocol parameters to be used.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
```

basic_seq_packet_socket::open (2 of 2 overloads)

Inherited from basic_socket.

Open the socket using the specified protocol.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code open(
    const protocol_type & protocol,
    boost::system::error_code & ec);
```

This function opens the socket so that it will use the specified protocol.

Parameters

protocol — An object specifying which protocol is to be used.

ec — Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::system::error_code ec;
socket.open(boost::asio::ip::tcp::v4(), ec);
if (ec)
{
  // An error occurred.
}
```

basic_seq_packet_socket::operator=

Move-assign a `basic_seq_packet_socket` from another.

```
basic_seq_packet_socket & operator=(
    basic_seq_packet_socket && other);
```

This assignment operator moves a sequenced packet socket from one object to another.

Parameters

other — The other `basic_seq_packet_socket` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_seq_packet_socket(io_service&) constructor`.

basic_seq_packet_socket::protocol_type

The protocol type.

```
typedef Protocol protocol_type;
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::receive

Receive some data on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags & out_flags);
  » more...

template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags in_flags,
    socket_base::message_flags & out_flags);
  » more...
```

Receive some data on a connected socket.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags in_flags,
    socket_base::message_flags & out_flags,
    boost::system::error_code & ec);
  » more...
```

basic_seq_packet_socket::receive (1 of 3 overloads)

Receive some data on the socket.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags & out_flags);
```

This function is used to receive data on the sequenced packet socket. The function call will block until data has been received successfully, or until an error occurs.

Parameters

buffers	One or more buffers into which the data will be received.
out_flags	After the receive call completes, contains flags associated with the received data. For example, if the `socket_base::message_end_of_record` bit is set then the received data marks the end of a record.

Return Value

The number of bytes received.

Exceptions

boost::system::system_error	Thrown on failure. An error code of `boost::asio::error::eof` indicates that the connection was closed by the peer.

Example

To receive into a single data buffer use the `buffer` function as follows:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
socket.receive(boost::asio::buffer(data, size), out_flags);
```

See the `buffer` documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_seq_packet_socket::receive (2 of 3 overloads)

Receive some data on the socket.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags in_flags,
    socket_base::message_flags & out_flags);
```

This function is used to receive data on the sequenced packet socket. The function call will block until data has been received successfully, or until an error occurs.

Parameters

buffers	One or more buffers into which the data will be received.
in_flags	Flags specifying how the receive call is to be made.
out_flags	After the receive call completes, contains flags associated with the received data. For example, if the `socket_base::message_end_of_record` bit is set then the received data marks the end of a record.

Return Value

The number of bytes received.

Exceptions

boost::system::system_error	Thrown on failure. An error code of `boost::asio::error::eof` indicates that the connection was closed by the peer.

Remarks

The receive operation may not receive all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

Example

To receive into a single data buffer use the `buffer` function as follows:

```
socket.receive(boost::asio::buffer(data, size), 0, out_flags);
```

See the `buffer` documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_seq_packet_socket::receive (3 of 3 overloads)

Receive some data on a connected socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags in_flags,
    socket_base::message_flags & out_flags,
    boost::system::error_code & ec);
```

This function is used to receive data on the sequenced packet socket. The function call will block until data has been received successfully, or until an error occurs.

Parameters

buffers	One or more buffers into which the data will be received.
in_flags	Flags specifying how the receive call is to be made.
out_flags	After the receive call completes, contains flags associated with the received data. For example, if the `socket_base::message_end_of_record` bit is set then the received data marks the end of a record.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes received. Returns 0 if an error occurred.

Remarks

The receive operation may not receive all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

basic_seq_packet_socket::receive_buffer_size

Inherited from socket_base.

Socket option for the receive buffer size of a socket.

```
typedef implementation_defined receive_buffer_size;
```

Implements the SOL_SOCKET/SO_RCVBUF socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option;
socket.get_option(option);
int size = option.value();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::receive_low_watermark

Inherited from socket_base.

Socket option for the receive low watermark.

```
typedef implementation_defined receive_low_watermark;
```

Implements the SOL_SOCKET/SO_RCVLOWAT socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::remote_endpoint

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint() const;
  » more...

endpoint_type remote_endpoint(
    boost::system::error_code & ec) const;
  » more...
```

basic_seq_packet_socket::remote_endpoint (1 of 2 overloads)

Inherited from basic_socket.

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint() const;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This function is used to obtain the remote endpoint of the socket.

Return Value

An object that represents the remote endpoint of the socket.

Exceptions

boost::system::system_error	Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::endpoint endpoint = socket.remote_endpoint();
```

basic_seq_packet_socket::remote_endpoint (2 of 2 overloads)

Inherited from basic_socket.

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint(
    boost::system::error_code & ec) const;
```

This function is used to obtain the remote endpoint of the socket.

Parameters

ec	Set to indicate what error occurred, if any.

Return Value

An object that represents the remote endpoint of the socket. Returns a default-constructed endpoint object if an error occurred.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
boost::asio::ip::tcp::endpoint endpoint = socket.remote_endpoint(ec);
if (ec)
{
  // An error occurred.
}
```

basic_seq_packet_socket::reuse_address

Inherited from socket_base.

Socket option to allow the socket to be bound to an address that is already in use.

```
typedef implementation_defined reuse_address;
```

Implements the SOL_SOCKET/SO_REUSEADDR socket option.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option(true);
acceptor.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::send

Send some data on the socket.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags);
  » more...

template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
  » more...
```

basic_seq_packet_socket::send (1 of 2 overloads)

Send some data on the socket.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags);
```

This function is used to send data on the sequenced packet socket. The function call will block until the data has been sent successfully, or an until error occurs.

Parameters

buffers One or more data buffers to be sent on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

flags Flags specifying how the send call is to be made.

Return Value

The number of bytes sent.

Exceptions

boost::system::system_error Thrown on failure.

Example

To send a single data buffer use the `buffer` function as follows:

```
socket.send(boost::asio::buffer(data, size), 0);
```

See the `buffer` documentation for information on sending multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_seq_packet_socket::send (2 of 2 overloads)

Send some data on the socket.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

This function is used to send data on the sequenced packet socket. The function call will block the data has been sent successfully, or an until error occurs.

Parameters

buffers One or more data buffers to be sent on the socket.

flags Flags specifying how the send call is to be made.

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes sent. Returns 0 if an error occurred.

Remarks

The send operation may not transmit all of the data to the peer. Consider using the `write` function if you need to ensure that all data is written before the blocking operation completes.

basic_seq_packet_socket::send_buffer_size

Inherited from socket_base.

Socket option for the send buffer size of a socket.

```
typedef implementation_defined send_buffer_size;
```

Implements the SOL_SOCKET/SO_SNDBUF socket option.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::send_low_watermark

Inherited from socket_base.

Socket option for the send low watermark.

```
typedef implementation_defined send_low_watermark;
```

Implements the SOL_SOCKET/SO_SNDLOWAT socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_seq_packet_socket::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

basic_seq_packet_socket::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef SeqPacketSocketService service_type;
```

Requirements

Header: `boost/asio/basic_seq_packet_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_seq_packet_socket::set_option

Set an option on the socket.

```
void set_option(
    const SettableSocketOption & option);
  » more...

boost::system::error_code set_option(
    const SettableSocketOption & option,
    boost::system::error_code & ec);
  » more...
```

basic_seq_packet_socket::set_option (1 of 2 overloads)

Inherited from basic_socket.

Set an option on the socket.

```
template<
    typename SettableSocketOption>
void set_option(
    const SettableSocketOption & option);
```

This function is used to set an option on the socket.

Parameters

option The new option value to be set on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Example

Setting the IPPROTO_TCP/TCP_NODELAY option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::no_delay option(true);
socket.set_option(option);
```

basic_seq_packet_socket::set_option (2 of 2 overloads)

Inherited from basic_socket.

Set an option on the socket.

```
template<
    typename SettableSocketOption>
boost::system::error_code set_option(
    const SettableSocketOption & option,
    boost::system::error_code & ec);
```

This function is used to set an option on the socket.

Parameters

option	The new option value to be set on the socket.
ec	Set to indicate what error occurred, if any.

Example

Setting the IPPROTO_TCP/TCP_NODELAY option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::no_delay option(true);
boost::system::error_code ec;
socket.set_option(option, ec);
if (ec)
{
  // An error occurred.
}
```

basic_seq_packet_socket::shutdown

Disable sends or receives on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void shutdown(
    shutdown_type what);
  » more...

boost::system::error_code shutdown(
    shutdown_type what,
    boost::system::error_code & ec);
  » more...
```

basic_seq_packet_socket::shutdown (1 of 2 overloads)

Inherited from basic_socket.

Disable sends or receives on the socket.

```
void shutdown(
    shutdown_type what);
```

This function is used to disable send operations, receive operations, or both.

Parameters

what Determines what types of operation will no longer be allowed.

Exceptions

boost::system::system_error Thrown on failure.

Example

Shutting down the send side of the socket:

```
boost::asio::ip::tcp::socket socket(io_service);
...
socket.shutdown(boost::asio::ip::tcp::socket::shutdown_send);
```

basic_seq_packet_socket::shutdown (2 of 2 overloads)

Inherited from basic_socket.

Disable sends or receives on the socket.

```
boost::system::error_code shutdown(
    shutdown_type what,
    boost::system::error_code & ec);
```

This function is used to disable send operations, receive operations, or both.

Parameters

what Determines what types of operation will no longer be allowed.

ec Set to indicate what error occurred, if any.

Example

Shutting down the send side of the socket:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
socket.shutdown(boost::asio::ip::tcp::socket::shutdown_send, ec);
if (ec)
{
  // An error occurred.
}
```

basic_seq_packet_socket::shutdown_type

Inherited from socket_base.

Different ways a socket may be shutdown.

```
enum shutdown_type
```

Values

shutdown_receive	Shutdown the receive side of the socket.
shutdown_send	Shutdown the send side of the socket.
shutdown_both	Shutdown both send and receive on the socket.

basic_serial_port

Provides serial port functionality.

```
template<
    typename SerialPortService = serial_port_service>
class basic_serial_port :
  public basic_io_object< SerialPortService >,
  public serial_port_base
```

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_serial_port is always the lowest layer.
native_handle_type	The native representation of a serial port.
native_type	(Deprecated: Use native_handle_type.) The native representation of a serial port.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native serial port to the serial port.
async_read_some	Start an asynchronous read.
async_write_some	Start an asynchronous write.
basic_serial_port	Construct a basic_serial_port without opening it. Construct and open a basic_serial_port. Construct a basic_serial_port on an existing native serial port. Move-construct a basic_serial_port from another.
cancel	Cancel all asynchronous operations associated with the serial port.
close	Close the serial port.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the serial port.
is_open	Determine whether the serial port is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native serial port representation.
native_handle	Get the native serial port representation.
open	Open the serial port using the specified device name.
operator=	Move-assign a basic_serial_port from another.
read_some	Read some data from the serial port.
send_break	Send a break sequence to the serial port.
set_option	Set an option on the serial port.
write_some	Write some data to the serial port.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_serial_port` class template provides functionality that is common to all serial ports.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/basic_serial_port.hpp`

Convenience header: `boost/asio.hpp`

basic_serial_port::assign

Assign an existing native serial port to the serial port.

```
void assign(
    const native_handle_type & native_serial_port);
  » more...

boost::system::error_code assign(
    const native_handle_type & native_serial_port,
    boost::system::error_code & ec);
  » more...
```

basic_serial_port::assign (1 of 2 overloads)

Assign an existing native serial port to the serial port.

```
void assign(
    const native_handle_type & native_serial_port);
```

basic_serial_port::assign (2 of 2 overloads)

Assign an existing native serial port to the serial port.

```
boost::system::error_code assign(
    const native_handle_type & native_serial_port,
    boost::system::error_code & ec);
```

basic_serial_port::async_read_some

Start an asynchronous read.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_some(
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

This function is used to asynchronously read data from the serial port. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be read. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes read.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The read operation may not read all of the requested number of bytes. Consider using the `async_read` function if you need to ensure that the requested amount of data is read before the asynchronous operation completes.

Example

To read into a single data buffer use the `buffer` function as follows:

```
serial_port.async_read_some(boost::asio::buffer(data, size), handler);
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_serial_port::async_write_some

Start an asynchronous write.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_some(
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

This function is used to asynchronously write data to the serial port. The function call always returns immediately.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

buffers — One or more data buffers to be written to the serial port. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the write operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes written.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The write operation may not transmit all of the data to the peer. Consider using the `async_write` function if you need to ensure that all data is written before the asynchronous operation completes.

Example

To write a single data buffer use the `buffer` function as follows:

```
serial_port.async_write_some(boost::asio::buffer(data, size), handler);
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_serial_port::basic_serial_port

Construct a `basic_serial_port` without opening it.

```
explicit basic_serial_port(
    boost::asio::io_service & io_service);
  » more...
```

Construct and open a `basic_serial_port`.

```
explicit basic_serial_port(
    boost::asio::io_service & io_service,
    const char * device);
  » more...

explicit basic_serial_port(
    boost::asio::io_service & io_service,
    const std::string & device);
  » more...
```

Construct a `basic_serial_port` on an existing native serial port.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
basic_serial_port(
    boost::asio::io_service & io_service,
    const native_handle_type & native_serial_port);
  » more...
```

Move-construct a basic_serial_port from another.

```
basic_serial_port(
    basic_serial_port && other);
  » more...
```

basic_serial_port::basic_serial_port (1 of 5 overloads)

Construct a basic_serial_port without opening it.

```
basic_serial_port(
    boost::asio::io_service & io_service);
```

This constructor creates a serial port without opening it.

Parameters

io_service — The io_service object that the serial port will use to dispatch handlers for any asynchronous operations performed on the port.

basic_serial_port::basic_serial_port (2 of 5 overloads)

Construct and open a basic_serial_port.

```
basic_serial_port(
    boost::asio::io_service & io_service,
    const char * device);
```

This constructor creates and opens a serial port for the specified device name.

Parameters

io_service — The io_service object that the serial port will use to dispatch handlers for any asynchronous operations performed on the port.

device — The platform-specific device name for this serial port.

basic_serial_port::basic_serial_port (3 of 5 overloads)

Construct and open a basic_serial_port.

```
basic_serial_port(
    boost::asio::io_service & io_service,
    const std::string & device);
```

This constructor creates and opens a serial port for the specified device name.

Parameters

io_service — The io_service object that the serial port will use to dispatch handlers for any asynchronous operations performed on the port.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

device The platform-specific device name for this serial port.

basic_serial_port::basic_serial_port (4 of 5 overloads)

Construct a `basic_serial_port` on an existing native serial port.

```
basic_serial_port(
    boost::asio::io_service & io_service,
    const native_handle_type & native_serial_port);
```

This constructor creates a serial port object to hold an existing native serial port.

Parameters

io_service The `io_service` object that the serial port will use to dispatch handlers for any asynchronous operations performed on the port.

native_serial_port A native serial port.

Exceptions

boost::system::system_error Thrown on failure.

basic_serial_port::basic_serial_port (5 of 5 overloads)

Move-construct a `basic_serial_port` from another.

```
basic_serial_port(
    basic_serial_port && other);
```

This constructor moves a serial port from one object to another.

Parameters

other The other `basic_serial_port` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_serial_port(io_service&)` constructor.

basic_serial_port::cancel

Cancel all asynchronous operations associated with the serial port.

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

basic_serial_port::cancel (1 of 2 overloads)

Cancel all asynchronous operations associated with the serial port.

```
void cancel();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This function causes all outstanding asynchronous read or write operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure.

basic_serial_port::cancel (2 of 2 overloads)

Cancel all asynchronous operations associated with the serial port.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function causes all outstanding asynchronous read or write operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Parameters

ec	Set to indicate what error occurred, if any.

basic_serial_port::close

Close the serial port.

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

basic_serial_port::close (1 of 2 overloads)

Close the serial port.

```
void close();
```

This function is used to close the serial port. Any asynchronous read or write operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure.

basic_serial_port::close (2 of 2 overloads)

Close the serial port.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```

This function is used to close the serial port. Any asynchronous read or write operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

ec Set to indicate what error occurred, if any.

basic_serial_port::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

basic_serial_port::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

basic_serial_port::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

basic_serial_port::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

basic_serial_port::get_option

Get an option from the serial port.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename GettableSerialPortOption>
void get_option(
    GettableSerialPortOption & option);
  » more...

template<
    typename GettableSerialPortOption>
boost::system::error_code get_option(
    GettableSerialPortOption & option,
    boost::system::error_code & ec);
  » more...
```

basic_serial_port::get_option (1 of 2 overloads)

Get an option from the serial port.

```
template<
    typename GettableSerialPortOption>
void get_option(
    GettableSerialPortOption & option);
```

This function is used to get the current value of an option on the serial port.

Parameters

option The option value to be obtained from the serial port.

Exceptions

boost::system::system_error Thrown on failure.

basic_serial_port::get_option (2 of 2 overloads)

Get an option from the serial port.

```
template<
    typename GettableSerialPortOption>
boost::system::error_code get_option(
    GettableSerialPortOption & option,
    boost::system::error_code & ec);
```

This function is used to get the current value of an option on the serial port.

Parameters

option The option value to be obtained from the serial port.

ec Set to indicate what error occured, if any.

basic_serial_port::get_service

Get the service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

basic_serial_port::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

basic_serial_port::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

basic_serial_port::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

basic_serial_port::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/basic_serial_port.hpp`

Convenience header: `boost/asio.hpp`

basic_serial_port::is_open

Determine whether the serial port is open.

```
bool is_open() const;
```

basic_serial_port::lowest_layer

Get a reference to the lowest layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

basic_serial_port::lowest_layer (1 of 2 overloads)

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

This function returns a reference to the lowest layer in a stack of layers. Since a `basic_serial_port` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

basic_serial_port::lowest_layer (2 of 2 overloads)

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

This function returns a const reference to the lowest layer in a stack of layers. Since a `basic_serial_port` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A const reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

basic_serial_port::lowest_layer_type

A `basic_serial_port` is always the lowest layer.

```
typedef basic_serial_port< SerialPortService > lowest_layer_type;
```

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_serial_port is always the lowest layer.
native_handle_type	The native representation of a serial port.
native_type	(Deprecated: Use native_handle_type.) The native representation of a serial port.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native serial port to the serial port.
async_read_some	Start an asynchronous read.
async_write_some	Start an asynchronous write.
basic_serial_port	Construct a basic_serial_port without opening it. Construct and open a basic_serial_port. Construct a basic_serial_port on an existing native serial port. Move-construct a basic_serial_port from another.
cancel	Cancel all asynchronous operations associated with the serial port.
close	Close the serial port.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the serial port.
is_open	Determine whether the serial port is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native serial port representation.
native_handle	Get the native serial port representation.
open	Open the serial port using the specified device name.
operator=	Move-assign a basic_serial_port from another.
read_some	Read some data from the serial port.
send_break	Send a break sequence to the serial port.
set_option	Set an option on the serial port.
write_some	Write some data to the serial port.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_serial_port` class template provides functionality that is common to all serial ports.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/basic_serial_port.hpp`

Convenience header: `boost/asio.hpp`

basic_serial_port::native

(Deprecated: Use `native_handle()`.) Get the native serial port representation.

```
native_type native();
```

This function may be used to obtain the underlying representation of the serial port. This is intended to allow access to native serial port functionality that is not otherwise provided.

basic_serial_port::native_handle

Get the native serial port representation.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying representation of the serial port. This is intended to allow access to native serial port functionality that is not otherwise provided.

basic_serial_port::native_handle_type

The native representation of a serial port.

```
typedef SerialPortService::native_handle_type native_handle_type;
```

Requirements

Header: `boost/asio/basic_serial_port.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_serial_port::native_type

(Deprecated: Use native_handle_type.) The native representation of a serial port.

```
typedef SerialPortService::native_handle_type native_type;
```

Requirements

Header: `boost/asio/basic_serial_port.hpp`

Convenience header: `boost/asio.hpp`

basic_serial_port::open

Open the serial port using the specified device name.

```
void open(
    const std::string & device);
  » more...

boost::system::error_code open(
    const std::string & device,
    boost::system::error_code & ec);
  » more...
```

basic_serial_port::open (1 of 2 overloads)

Open the serial port using the specified device name.

```
void open(
    const std::string & device);
```

This function opens the serial port for the specified device name.

Parameters

device The platform-specific device name.

Exceptions

boost::system::system_error Thrown on failure.

basic_serial_port::open (2 of 2 overloads)

Open the serial port using the specified device name.

```
boost::system::error_code open(
    const std::string & device,
    boost::system::error_code & ec);
```

This function opens the serial port using the given platform-specific device name.

Parameters

device The platform-specific device name.

ec Set the indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_serial_port::operator=

Move-assign a `basic_serial_port` from another.

```
basic_serial_port & operator=(
    basic_serial_port && other);
```

This assignment operator moves a serial port from one object to another.

Parameters

other The other `basic_serial_port` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_serial_port(io_service&)` constructor.

basic_serial_port::read_some

Read some data from the serial port.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
  » more...

template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

basic_serial_port::read_some (1 of 2 overloads)

Read some data from the serial port.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
```

This function is used to read data from the serial port. The function call will block until one or more bytes of data has been read successfully, or until an error occurs.

Parameters

buffers One or more buffers into which the data will be read.

Return Value

The number of bytes read.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure. An error code of `boost::asio::error::eof` indicates that the connection was closed by the peer.

Remarks

The read_some operation may not read all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

Example

To read into a single data buffer use the `buffer` function as follows:

```
serial_port.read_some(boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_serial_port::read_some (2 of 2 overloads)

Read some data from the serial port.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to read data from the serial port. The function call will block until one or more bytes of data has been read successfully, or until an error occurs.

Parameters

buffers	One or more buffers into which the data will be read.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes read. Returns 0 if an error occurred.

Remarks

The read_some operation may not read all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

basic_serial_port::send_break

Send a break sequence to the serial port.

```
void send_break();
  » more...

boost::system::error_code send_break(
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_serial_port::send_break (1 of 2 overloads)

Send a break sequence to the serial port.

```
void send_break();
```

This function causes a break sequence of platform-specific duration to be sent out the serial port.

Exceptions

boost::system::system_error Thrown on failure.

basic_serial_port::send_break (2 of 2 overloads)

Send a break sequence to the serial port.

```
boost::system::error_code send_break(
    boost::system::error_code & ec);
```

This function causes a break sequence of platform-specific duration to be sent out the serial port.

Parameters

ec Set to indicate what error occurred, if any.

basic_serial_port::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

basic_serial_port::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef SerialPortService service_type;
```

Requirements

Header: `boost/asio/basic_serial_port.hpp`

Convenience header: `boost/asio.hpp`

basic_serial_port::set_option

Set an option on the serial port.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename SettableSerialPortOption>
void set_option(
    const SettableSerialPortOption & option);
  » more...

template<
    typename SettableSerialPortOption>
boost::system::error_code set_option(
    const SettableSerialPortOption & option,
    boost::system::error_code & ec);
  » more...
```

basic_serial_port::set_option (1 of 2 overloads)

Set an option on the serial port.

```
template<
    typename SettableSerialPortOption>
void set_option(
    const SettableSerialPortOption & option);
```

This function is used to set an option on the serial port.

Parameters

option The option value to be set on the serial port.

Exceptions

boost::system::system_error Thrown on failure.

basic_serial_port::set_option (2 of 2 overloads)

Set an option on the serial port.

```
template<
    typename SettableSerialPortOption>
boost::system::error_code set_option(
    const SettableSerialPortOption & option,
    boost::system::error_code & ec);
```

This function is used to set an option on the serial port.

Parameters

option The option value to be set on the serial port.

ec Set to indicate what error occurred, if any.

basic_serial_port::write_some

Write some data to the serial port.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
  » more...

template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

basic_serial_port::write_some (1 of 2 overloads)

Write some data to the serial port.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
```

This function is used to write data to the serial port. The function call will block until one or more bytes of the data has been written successfully, or until an error occurs.

Parameters

buffers — One or more data buffers to be written to the serial port.

Return Value

The number of bytes written.

Exceptions

boost::system::system_error — Thrown on failure. An error code of `boost::asio::error::eof` indicates that the connection was closed by the peer.

Remarks

The write_some operation may not transmit all of the data to the peer. Consider using the `write` function if you need to ensure that all data is written before the blocking operation completes.

Example

To write a single data buffer use the `buffer` function as follows:

```
serial_port.write_some(boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_serial_port::write_some (2 of 2 overloads)

Write some data to the serial port.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to write data to the serial port. The function call will block until one or more bytes of the data has been written successfully, or until an error occurs.

Parameters

buffers One or more data buffers to be written to the serial port.

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes written. Returns 0 if an error occurred.

Remarks

The write_some operation may not transmit all of the data to the peer. Consider using the `write` function if you need to ensure that all data is written before the blocking operation completes.

basic_signal_set

Provides signal functionality.

```
template<
    typename SignalSetService = signal_set_service>
class basic_signal_set :
  public basic_io_object< SignalSetService >
```

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
add	Add a signal to a signal_set.
async_wait	Start an asynchronous operation to wait for a signal to be delivered.
basic_signal_set	Construct a signal set without adding any signals. Construct a signal set and add one signal. Construct a signal set and add two signals. Construct a signal set and add three signals.
cancel	Cancel all operations associated with the signal set.
clear	Remove all signals from a signal_set.
get_io_service	Get the io_service associated with the object.
remove	Remove a signal from a signal_set.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_signal_set` class template provides the ability to perform an asynchronous wait for one or more signals to occur.

Most applications will use the `signal_set` typedef.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Example

Performing an asynchronous wait:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void handler(
    const boost::system::error_code& error,
    int signal_number)
{
  if (!error)
  {
    // A signal occurred.
  }
}

...

// Construct a signal set registered for process termination.
boost::asio::signal_set signals(io_service, SIGINT, SIGTERM);

// Start an asynchronous wait for one of the signals to occur.
signals.async_wait(handler);
```

Queueing of signal notifications

If a signal is registered with a signal_set, and the signal occurs when there are no waiting handlers, then the signal notification is queued. The next async_wait operation on that signal_set will dequeue the notification. If multiple notifications are queued, subsequent async_wait operations dequeue them one at a time. Signal notifications are dequeued in order of ascending signal number.

If a signal number is removed from a signal_set (using the `remove` or `erase` member functions) then any queued notifications for that signal are discarded.

Multiple registration of signals

The same signal number may be registered with different signal_set objects. When the signal occurs, one handler is called for each signal_set object.

Note that multiple registration only works for signals that are registered using Asio. The application must not also register a signal handler using functions such as `signal()` or `sigaction()`.

Signal masking on POSIX platforms

POSIX allows signals to be blocked using functions such as `sigprocmask()` and `pthread_sigmask()`. For signals to be delivered, programs must ensure that any signals registered using signal_set objects are unblocked in at least one thread.

Requirements

Header: `boost/asio/basic_signal_set.hpp`

Convenience header: `boost/asio.hpp`

basic_signal_set::add

Add a signal to a signal_set.

```
void add(
    int signal_number);
  » more...

boost::system::error_code add(
    int signal_number,
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_signal_set::add (1 of 2 overloads)

Add a signal to a signal_set.

```
void add(
    int signal_number);
```

This function adds the specified signal to the set. It has no effect if the signal is already in the set.

Parameters

signal_number The signal to be added to the set.

Exceptions

boost::system::system_error Thrown on failure.

basic_signal_set::add (2 of 2 overloads)

Add a signal to a signal_set.

```
boost::system::error_code add(
    int signal_number,
    boost::system::error_code & ec);
```

This function adds the specified signal to the set. It has no effect if the signal is already in the set.

Parameters

signal_number The signal to be added to the set.

ec Set to indicate what error occurred, if any.

basic_signal_set::async_wait

Start an asynchronous operation to wait for a signal to be delivered.

```
template<
    typename SignalHandler>
void async_wait(
    SignalHandler handler);
```

This function may be used to initiate an asynchronous wait against the signal set. It always returns immediately.

For each call to `async_wait()`, the supplied handler will be called exactly once. The handler will be called when:

- One of the registered signals in the signal set occurs; or
- The signal set was cancelled, in which case the handler is passed the error code `boost::asio::error::operation_aborted`.

Parameters

handler The handler to be called when the signal occurs. Copies will be made of the handler as required. The function signature of the handler must be:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  int signal_number // Indicates which signal occurred.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

basic_signal_set::basic_signal_set

Construct a signal set without adding any signals.

```
explicit basic_signal_set(
    boost::asio::io_service & io_service);
  » more...
```

Construct a signal set and add one signal.

```
basic_signal_set(
    boost::asio::io_service & io_service,
    int signal_number_1);
  » more...
```

Construct a signal set and add two signals.

```
basic_signal_set(
    boost::asio::io_service & io_service,
    int signal_number_1,
    int signal_number_2);
  » more...
```

Construct a signal set and add three signals.

```
basic_signal_set(
    boost::asio::io_service & io_service,
    int signal_number_1,
    int signal_number_2,
    int signal_number_3);
  » more...
```

basic_signal_set::basic_signal_set (1 of 4 overloads)

Construct a signal set without adding any signals.

```
basic_signal_set(
    boost::asio::io_service & io_service);
```

This constructor creates a signal set without registering for any signals.

Parameters

io_service	The `io_service` object that the signal set will use to dispatch handlers for any asynchronous operations performed on the set.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_signal_set::basic_signal_set (2 of 4 overloads)

Construct a signal set and add one signal.

```
basic_signal_set(
    boost::asio::io_service & io_service,
    int signal_number_1);
```

This constructor creates a signal set and registers for one signal.

Parameters

io_service	The `io_service` object that the signal set will use to dispatch handlers for any asynchronous operations performed on the set.
signal_number_1	The signal number to be added.

Remarks

This constructor is equivalent to performing:

```
boost::asio::signal_set signals(io_service);
signals.add(signal_number_1);
```

basic_signal_set::basic_signal_set (3 of 4 overloads)

Construct a signal set and add two signals.

```
basic_signal_set(
    boost::asio::io_service & io_service,
    int signal_number_1,
    int signal_number_2);
```

This constructor creates a signal set and registers for two signals.

Parameters

io_service	The `io_service` object that the signal set will use to dispatch handlers for any asynchronous operations performed on the set.
signal_number_1	The first signal number to be added.
signal_number_2	The second signal number to be added.

Remarks

This constructor is equivalent to performing:

```
boost::asio::signal_set signals(io_service);
signals.add(signal_number_1);
signals.add(signal_number_2);
```

basic_signal_set::basic_signal_set (4 of 4 overloads)

Construct a signal set and add three signals.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
basic_signal_set(
    boost::asio::io_service & io_service,
    int signal_number_1,
    int signal_number_2,
    int signal_number_3);
```

This constructor creates a signal set and registers for three signals.

Parameters

io_service	The `io_service` object that the signal set will use to dispatch handlers for any asynchronous operations performed on the set.
signal_number_1	The first signal number to be added.
signal_number_2	The second signal number to be added.
signal_number_3	The third signal number to be added.

Remarks

This constructor is equivalent to performing:

```
boost::asio::signal_set signals(io_service);
signals.add(signal_number_1);
signals.add(signal_number_2);
signals.add(signal_number_3);
```

basic_signal_set::cancel

Cancel all operations associated with the signal set.

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

basic_signal_set::cancel (1 of 2 overloads)

Cancel all operations associated with the signal set.

```
void cancel();
```

This function forces the completion of any pending asynchronous wait operations against the signal set. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Cancellation does not alter the set of registered signals.

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

If a registered signal occurred before `cancel()` is called, then the handlers for asynchronous wait operations will:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_signal_set::cancel (2 of 2 overloads)

Cancel all operations associated with the signal set.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function forces the completion of any pending asynchronous wait operations against the signal set. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Cancellation does not alter the set of registered signals.

Parameters

ec Set to indicate what error occurred, if any.

Remarks

If a registered signal occurred before `cancel()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_signal_set::clear

Remove all signals from a signal_set.

```
void clear();
  » more...

boost::system::error_code clear(
    boost::system::error_code & ec);
  » more...
```

basic_signal_set::clear (1 of 2 overloads)

Remove all signals from a signal_set.

```
void clear();
```

This function removes all signals from the set. It has no effect if the set is already empty.

Exceptions

boost::system::system_error Thrown on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

Removes all queued notifications.

basic_signal_set::clear (2 of 2 overloads)

Remove all signals from a signal_set.

```
boost::system::error_code clear(
    boost::system::error_code & ec);
```

This function removes all signals from the set. It has no effect if the set is already empty.

Parameters

ec Set to indicate what error occurred, if any.

Remarks

Removes all queued notifications.

basic_signal_set::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

basic_signal_set::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

basic_signal_set::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

basic_signal_set::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

basic_signal_set::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

basic_signal_set::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

basic_signal_set::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

basic_signal_set::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

basic_signal_set::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/basic_signal_set.hpp`

Convenience header: `boost/asio.hpp`

basic_signal_set::remove

Remove a signal from a signal_set.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void remove(
    int signal_number);
  » more...

boost::system::error_code remove(
    int signal_number,
    boost::system::error_code & ec);
  » more...
```

basic_signal_set::remove (1 of 2 overloads)

Remove a signal from a signal_set.

```
void remove(
    int signal_number);
```

This function removes the specified signal from the set. It has no effect if the signal is not in the set.

Parameters

signal_number The signal to be removed from the set.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

Removes any notifications that have been queued for the specified signal number.

basic_signal_set::remove (2 of 2 overloads)

Remove a signal from a signal_set.

```
boost::system::error_code remove(
    int signal_number,
    boost::system::error_code & ec);
```

This function removes the specified signal from the set. It has no effect if the signal is not in the set.

Parameters

signal_number The signal to be removed from the set.

ec Set to indicate what error occurred, if any.

Remarks

Removes any notifications that have been queued for the specified signal number.

basic_signal_set::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

Available only for services that do not support movability.

basic_signal_set::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef SignalSetService service_type;
```

Requirements

Header: `boost/asio/basic_signal_set.hpp`

Convenience header: `boost/asio.hpp`

basic_socket

Provides socket functionality.

```
template<
    typename Protocol,
    typename SocketService>
class basic_socket :
  public basic_io_object< SocketService >,
  public socket_base
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_socket	Construct a basic_socket without opening it. Construct and open a basic_socket. Construct a basic_socket, opening it and binding it to the given local endpoint. Construct a basic_socket on an existing native socket. Move-construct a basic_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_socket from another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
remote_endpoint	Get the remote endpoint of the socket.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_socket	Protected destructor to prevent deletion through this type.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_socket` class template provides functionality that is common to both stream-oriented and datagram-oriented sockets.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/basic_socket.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio.hpp`

basic_socket::assign

Assign an existing native socket to the socket.

```
void assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket);
  » more...

boost::system::error_code assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
  » more...
```

basic_socket::assign (1 of 2 overloads)

Assign an existing native socket to the socket.

```
void assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket);
```

basic_socket::assign (2 of 2 overloads)

Assign an existing native socket to the socket.

```
boost::system::error_code assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
```

basic_socket::async_connect

Start an asynchronous connect.

```
template<
    typename ConnectHandler>
void async_connect(
    const endpoint_type & peer_endpoint,
    ConnectHandler handler);
```

This function is used to asynchronously connect a socket to the specified remote endpoint. The function call always returns immediately.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint	The remote endpoint to which the socket will be connected. Copies will be made of the endpoint object as required.
handler	The handler to be called when the connection operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void handler(
  const boost::system::error_code& error // Result of operation
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

```
void connect_handler(const boost::system::error_code& error)
{
  if (!error)
  {
    // Connect succeeded.
  }
}

...

boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.async_connect(endpoint, connect_handler);
```

basic_socket::at_mark

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark() const;
  » more...

bool at_mark(
    boost::system::error_code & ec) const;
  » more...
```

basic_socket::at_mark (1 of 2 overloads)

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark() const;
```

This function is used to check whether the socket input is currently positioned at the out-of-band data mark.

Return Value

A bool indicating whether the socket is at the out-of-band data mark.

Exceptions

boost::system::system_error	Thrown on failure.

basic_socket::at_mark (2 of 2 overloads)

Determine whether the socket is at the out-of-band data mark.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
bool at_mark(
    boost::system::error_code & ec) const;
```

This function is used to check whether the socket input is currently positioned at the out-of-band data mark.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

A bool indicating whether the socket is at the out-of-band data mark.

basic_socket::available

Determine the number of bytes available for reading.

```
std::size_t available() const;
  » more...

std::size_t available(
    boost::system::error_code & ec) const;
  » more...
```

basic_socket::available (1 of 2 overloads)

Determine the number of bytes available for reading.

```
std::size_t available() const;
```

This function is used to determine the number of bytes that may be read without blocking.

Return Value

The number of bytes that may be read without blocking, or 0 if an error occurs.

Exceptions

boost::system::system_error Thrown on failure.

basic_socket::available (2 of 2 overloads)

Determine the number of bytes available for reading.

```
std::size_t available(
    boost::system::error_code & ec) const;
```

This function is used to determine the number of bytes that may be read without blocking.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes that may be read without blocking, or 0 if an error occurs.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket::basic_socket

Construct a basic_socket without opening it.

```
explicit basic_socket(
    boost::asio::io_service & io_service);
  » more...
```

Construct and open a basic_socket.

```
basic_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol);
  » more...
```

Construct a basic_socket, opening it and binding it to the given local endpoint.

```
basic_socket(
    boost::asio::io_service & io_service,
    const endpoint_type & endpoint);
  » more...
```

Construct a basic_socket on an existing native socket.

```
basic_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol,
    const native_handle_type & native_socket);
  » more...
```

Move-construct a basic_socket from another.

```
basic_socket(
    basic_socket && other);
  » more...
```

basic_socket::basic_socket (1 of 5 overloads)

Construct a basic_socket without opening it.

```
basic_socket(
    boost::asio::io_service & io_service);
```

This constructor creates a socket without opening it.

Parameters

io_service	The io_service object that the socket will use to dispatch handlers for any asynchronous operations performed on the socket.

basic_socket::basic_socket (2 of 5 overloads)

Construct and open a basic_socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
basic_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol);
```

This constructor creates and opens a socket.

Parameters

io_service	The `io_service` object that the socket will use to dispatch handlers for any asynchronous operations performed on the socket.
protocol	An object specifying protocol parameters to be used.

Exceptions

boost::system::system_error	Thrown on failure.

basic_socket::basic_socket (3 of 5 overloads)

Construct a `basic_socket`, opening it and binding it to the given local endpoint.

```
basic_socket(
    boost::asio::io_service & io_service,
    const endpoint_type & endpoint);
```

This constructor creates a socket and automatically opens it bound to the specified endpoint on the local machine. The protocol used is the protocol associated with the given endpoint.

Parameters

io_service	The `io_service` object that the socket will use to dispatch handlers for any asynchronous operations performed on the socket.
endpoint	An endpoint on the local machine to which the socket will be bound.

Exceptions

boost::system::system_error	Thrown on failure.

basic_socket::basic_socket (4 of 5 overloads)

Construct a `basic_socket` on an existing native socket.

```
basic_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol,
    const native_handle_type & native_socket);
```

This constructor creates a socket object to hold an existing native socket.

Parameters

io_service	The `io_service` object that the socket will use to dispatch handlers for any asynchronous operations performed on the socket.
protocol	An object specifying protocol parameters to be used.
native_socket	A native socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

basic_socket::basic_socket (5 of 5 overloads)

Move-construct a `basic_socket` from another.

```
basic_socket(
    basic_socket && other);
```

This constructor moves a socket from one object to another.

Parameters

other	The other `basic_socket` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_socket(io_service&)` constructor.

basic_socket::bind

Bind the socket to the given local endpoint.

```
void bind(
    const endpoint_type & endpoint);
  » more...

boost::system::error_code bind(
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
  » more...
```

basic_socket::bind (1 of 2 overloads)

Bind the socket to the given local endpoint.

```
void bind(
    const endpoint_type & endpoint);
```

This function binds the socket to the specified endpoint on the local machine.

Parameters

endpoint	An endpoint on the local machine to which the socket will be bound.

Exceptions

boost::system::system_error	Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
socket.bind(boost::asio::ip::tcp::endpoint(
      boost::asio::ip::tcp::v4(), 12345));
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket::bind (2 of 2 overloads)

Bind the socket to the given local endpoint.

```
boost::system::error_code bind(
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
```

This function binds the socket to the specified endpoint on the local machine.

Parameters

endpoint — An endpoint on the local machine to which the socket will be bound.

ec — Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
boost::system::error_code ec;
socket.bind(boost::asio::ip::tcp::endpoint(
      boost::asio::ip::tcp::v4(), 12345), ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket::broadcast

Inherited from socket_base.

Socket option to permit sending of broadcast messages.

```
typedef implementation_defined broadcast;
```

Implements the SOL_SOCKET/SO_BROADCAST socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio.hpp`

basic_socket::bytes_readable

Inherited from socket_base.

IO control command to get the amount of data that can be read without blocking.

```
typedef implementation_defined bytes_readable;
```

Implements the FIONREAD IO control command.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::bytes_readable command(true);
socket.io_control(command);
std::size_t bytes_readable = command.get();
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::cancel

Cancel all asynchronous operations associated with the socket.

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

basic_socket::cancel (1 of 2 overloads)

Cancel all asynchronous operations associated with the socket.

```
void cancel();
```

This function causes all outstanding asynchronous connect, send and receive operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

Calls to `cancel()` will always fail with `boost::asio::error::operation_not_supported` when run on Windows XP, Windows Server 2003, and earlier versions of Windows, unless BOOST_ASIO_ENABLE_CANCELIO is defined. However, the CancelIo function has two issues that should be considered before enabling its use:

- It will only cancel asynchronous operations that were initiated in the current thread.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- It can appear to complete without error, but the request to cancel the unfinished operations may be silently ignored by the operating system. Whether it works or not seems to depend on the drivers that are installed.

For portable cancellation, consider using one of the following alternatives:

- Disable asio's I/O completion port backend by defining BOOST_ASIO_DISABLE_IOCP.
- Use the `close()` function to simultaneously cancel the outstanding operations and close the socket.

When running on Windows Vista, Windows Server 2008, and later, the CancelIoEx function is always used. This function does not have the problems described above.

basic_socket::cancel (2 of 2 overloads)

Cancel all asynchronous operations associated with the socket.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function causes all outstanding asynchronous connect, send and receive operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Parameters

ec Set to indicate what error occurred, if any.

Remarks

Calls to `cancel()` will always fail with `boost::asio::error::operation_not_supported` when run on Windows XP, Windows Server 2003, and earlier versions of Windows, unless BOOST_ASIO_ENABLE_CANCELIO is defined. However, the CancelIo function has two issues that should be considered before enabling its use:

- It will only cancel asynchronous operations that were initiated in the current thread.
- It can appear to complete without error, but the request to cancel the unfinished operations may be silently ignored by the operating system. Whether it works or not seems to depend on the drivers that are installed.

For portable cancellation, consider using one of the following alternatives:

- Disable asio's I/O completion port backend by defining BOOST_ASIO_DISABLE_IOCP.
- Use the `close()` function to simultaneously cancel the outstanding operations and close the socket.

When running on Windows Vista, Windows Server 2008, and later, the CancelIoEx function is always used. This function does not have the problems described above.

basic_socket::close

Close the socket.

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket::close (1 of 2 overloads)

Close the socket.

```
void close();
```

This function is used to close the socket. Any asynchronous send, receive or connect operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure. Note that, even if the function indicates an error, the underlying descriptor is closed.

Remarks

For portable behaviour with respect to graceful closure of a connected socket, call `shutdown()` before closing the socket.

basic_socket::close (2 of 2 overloads)

Close the socket.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```

This function is used to close the socket. Any asynchronous send, receive or connect operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Parameters

ec	Set to indicate what error occurred, if any. Note that, even if the function indicates an error, the underlying descriptor is closed.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
socket.close(ec);
if (ec)
{
  // An error occurred.
}
```

Remarks

For portable behaviour with respect to graceful closure of a connected socket, call `shutdown()` before closing the socket.

basic_socket::connect

Connect the socket to the specified endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void connect(
    const endpoint_type & peer_endpoint);
  » more...

boost::system::error_code connect(
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
  » more...
```

basic_socket::connect (1 of 2 overloads)

Connect the socket to the specified endpoint.

```
void connect(
    const endpoint_type & peer_endpoint);
```

This function is used to connect a socket to the specified remote endpoint. The function call will block until the connection is successfully made or an error occurs.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint	The remote endpoint to which the socket will be connected.

Exceptions

boost::system::system_error	Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.connect(endpoint);
```

basic_socket::connect (2 of 2 overloads)

Connect the socket to the specified endpoint.

```
boost::system::error_code connect(
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
```

This function is used to connect a socket to the specified remote endpoint. The function call will block until the connection is successfully made or an error occurs.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint	The remote endpoint to which the socket will be connected.
ec	Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
boost::system::error_code ec;
socket.connect(endpoint, ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket::debug

Inherited from socket_base.

Socket option to enable socket-level debugging.

```
typedef implementation_defined debug;
```

Implements the SOL_SOCKET/SO_DEBUG socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::do_not_route

Inherited from socket_base.

Socket option to prevent routing, use local interfaces only.

```
typedef implementation_defined do_not_route;
```

Implements the SOL_SOCKET/SO_DONTROUTE socket option.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::enable_connection_aborted

Inherited from socket_base.

Socket option to report aborted connections on accept.

```
typedef implementation_defined enable_connection_aborted;
```

Implements a custom socket option that determines whether or not an accept operation is permitted to fail with `boost::asio::error::connection_aborted`. By default the option is false.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option(true);
acceptor.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket::endpoint_type

The endpoint type.

```
typedef Protocol::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

basic_socket::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

basic_socket::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

basic_socket::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

basic_socket::get_option

Get an option from the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename GettableSocketOption>
void get_option(
    GettableSocketOption & option) const;
  » more...

template<
    typename GettableSocketOption>
boost::system::error_code get_option(
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
  » more...
```

basic_socket::get_option (1 of 2 overloads)

Get an option from the socket.

```
template<
    typename GettableSocketOption>
void get_option(
    GettableSocketOption & option) const;
```

This function is used to get the current value of an option on the socket.

Parameters

option The option value to be obtained from the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Getting the value of the SOL_SOCKET/SO_KEEPALIVE option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::keep_alive option;
socket.get_option(option);
bool is_set = option.get();
```

basic_socket::get_option (2 of 2 overloads)

Get an option from the socket.

```
template<
    typename GettableSocketOption>
boost::system::error_code get_option(
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
```

This function is used to get the current value of an option on the socket.

Parameters

option The option value to be obtained from the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ec Set to indicate what error occurred, if any.

Example

Getting the value of the SOL_SOCKET/SO_KEEPALIVE option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::keep_alive option;
boost::system::error_code ec;
socket.get_option(option, ec);
if (ec)
{
  // An error occurred.
}
bool is_set = option.get();
```

basic_socket::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

basic_socket::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

basic_socket::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

basic_socket::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

basic_socket::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::io_control

Perform an IO control command on the socket.

```
template<
    typename IoControlCommand>
void io_control(
    IoControlCommand & command);
  » more...

template<
    typename IoControlCommand>
boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
  » more...
```

basic_socket::io_control (1 of 2 overloads)

Perform an IO control command on the socket.

```
template<
    typename IoControlCommand>
void io_control(
    IoControlCommand & command);
```

This function is used to execute an IO control command on the socket.

Parameters

command The IO control command to be performed on the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Getting the number of bytes ready to read:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::bytes_readable command;
socket.io_control(command);
std::size_t bytes_readable = command.get();
```

basic_socket::io_control (2 of 2 overloads)

Perform an IO control command on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
```

This function is used to execute an IO control command on the socket.

Parameters

command The IO control command to be performed on the socket.

ec Set to indicate what error occurred, if any.

Example

Getting the number of bytes ready to read:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::bytes_readable command;
boost::system::error_code ec;
socket.io_control(command, ec);
if (ec)
{
  // An error occurred.
}
std::size_t bytes_readable = command.get();
```

basic_socket::is_open

Determine whether the socket is open.

```
bool is_open() const;
```

basic_socket::keep_alive

Inherited from socket_base.

Socket option to send keep-alives.

```
typedef implementation_defined keep_alive;
```

Implements the SOL_SOCKET/SO_KEEPALIVE socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option(true);
socket.set_option(option);
```

Getting the current option value:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::linger

Inherited from socket_base.

Socket option to specify whether the socket lingers on close if unsent data is present.

```
typedef implementation_defined linger;
```

Implements the SOL_SOCKET/SO_LINGER socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option(true, 30);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option;
socket.get_option(option);
bool is_set = option.enabled();
unsigned short timeout = option.timeout();
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::local_endpoint

Get the local endpoint of the socket.

```
endpoint_type local_endpoint() const;
  » more...

endpoint_type local_endpoint(
    boost::system::error_code & ec) const;
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket::local_endpoint (1 of 2 overloads)

Get the local endpoint of the socket.

```
endpoint_type local_endpoint() const;
```

This function is used to obtain the locally bound endpoint of the socket.

Return Value

An object that represents the local endpoint of the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::endpoint endpoint = socket.local_endpoint();
```

basic_socket::local_endpoint (2 of 2 overloads)

Get the local endpoint of the socket.

```
endpoint_type local_endpoint(
    boost::system::error_code & ec) const;
```

This function is used to obtain the locally bound endpoint of the socket.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

An object that represents the local endpoint of the socket. Returns a default-constructed endpoint object if an error occurred.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
boost::asio::ip::tcp::endpoint endpoint = socket.local_endpoint(ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket::lowest_layer

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

basic_socket::lowest_layer (1 of 2 overloads)

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

This function returns a reference to the lowest layer in a stack of layers. Since a `basic_socket` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

basic_socket::lowest_layer (2 of 2 overloads)

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

This function returns a const reference to the lowest layer in a stack of layers. Since a `basic_socket` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A const reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

basic_socket::lowest_layer_type

A `basic_socket` is always the lowest layer.

```
typedef basic_socket< Protocol, SocketService > lowest_layer_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_socket	Construct a basic_socket without opening it. Construct and open a basic_socket. Construct a basic_socket, opening it and binding it to the given local endpoint. Construct a basic_socket on an existing native socket. Move-construct a basic_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_socket from another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
remote_endpoint	Get the remote endpoint of the socket.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_socket	Protected destructor to prevent deletion through this type.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_socket` class template provides functionality that is common to both stream-oriented and datagram-oriented sockets.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket::max_connections

Inherited from socket_base.

The maximum length of the queue of pending incoming connections.

```
static const int max_connections = implementation_defined;
```

basic_socket::message_do_not_route

Inherited from socket_base.

Specify that the data should not be subject to routing.

```
static const int message_do_not_route = implementation_defined;
```

basic_socket::message_end_of_record

Inherited from socket_base.

Specifies that the data marks the end of a record.

```
static const int message_end_of_record = implementation_defined;
```

basic_socket::message_flags

Inherited from socket_base.

Bitmask type for flags that can be passed to send and receive operations.

```
typedef int message_flags;
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::message_out_of_band

Inherited from socket_base.

Process out-of-band data.

```
static const int message_out_of_band = implementation_defined;
```

basic_socket::message_peek

Inherited from socket_base.

Peek at incoming data without removing it from the input queue.

```
static const int message_peek = implementation_defined;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket::native

(Deprecated: Use `native_handle()`.) Get the native socket representation.

```
native_type native();
```

This function may be used to obtain the underlying representation of the socket. This is intended to allow access to native socket functionality that is not otherwise provided.

basic_socket::native_handle

Get the native socket representation.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying representation of the socket. This is intended to allow access to native socket functionality that is not otherwise provided.

basic_socket::native_handle_type

The native representation of a socket.

```
typedef SocketService::native_handle_type native_handle_type;
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::native_non_blocking

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking() const;
  » more...
```

Sets the non-blocking mode of the native socket implementation.

```
void native_non_blocking(
    bool mode);
  » more...

boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

basic_socket::native_non_blocking (1 of 3 overloads)

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking() const;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This function is used to retrieve the non-blocking mode of the underlying native socket. This mode has no effect on the behaviour of the socket object's synchronous operations.

Return Value

`true` if the underlying socket is in non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Remarks

The current non-blocking mode is cached by the socket object. Consequently, the return value may be incorrect if the non-blocking mode was set directly on the native socket.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};

template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_socket::native_non_blocking (2 of 3 overloads)

Sets the non-blocking mode of the native socket implementation.

```
void native_non_blocking(
    bool mode);
```

This function is used to modify the non-blocking mode of the underlying native socket. It has no effect on the behaviour of the socket object's synchronous operations.

Parameters

mode — If `true`, the underlying socket is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Exceptions

boost::system::system_error — Thrown on failure. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_socket::native_non_blocking (3 of 3 overloads)

Sets the non-blocking mode of the native socket implementation.

```
boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

This function is used to modify the non-blocking mode of the underlying native socket. It has no effect on the behaviour of the socket object's synchronous operations.

Parameters

mode — If `true`, the underlying socket is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

ec — Set to indicate what error occurred, if any. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_socket::native_type

(Deprecated: Use native_handle_type.) The native representation of a socket.

```
typedef SocketService::native_handle_type native_type;
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::non_blocking

Gets the non-blocking mode of the socket.

```
bool non_blocking() const;
  » more...
```

Sets the non-blocking mode of the socket.

```
void non_blocking(
    bool mode);
  » more...

boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

basic_socket::non_blocking (1 of 3 overloads)

Gets the non-blocking mode of the socket.

```
bool non_blocking() const;
```

Return Value

`true` if the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_socket::non_blocking (2 of 3 overloads)

Sets the non-blocking mode of the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void non_blocking(
    bool mode);
```

Parameters

mode — If `true`, the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Exceptions

boost::system::system_error — Thrown on failure.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_socket::non_blocking (3 of 3 overloads)

Sets the non-blocking mode of the socket.

```
boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

Parameters

mode — If `true`, the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

ec — Set to indicate what error occurred, if any.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_socket::non_blocking_io

Inherited from socket_base.

(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.

```
typedef implementation_defined non_blocking_io;
```

Implements the FIONBIO IO control command.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::non_blocking_io command(true);
socket.io_control(command);
```

Requirements

Header: `boost/asio/basic_socket.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio.hpp`

basic_socket::open

Open the socket using the specified protocol.

```
void open(
    const protocol_type & protocol = protocol_type());
  » more...

boost::system::error_code open(
    const protocol_type & protocol,
    boost::system::error_code & ec);
  » more...
```

basic_socket::open (1 of 2 overloads)

Open the socket using the specified protocol.

```
void open(
    const protocol_type & protocol = protocol_type());
```

This function opens the socket so that it will use the specified protocol.

Parameters

protocol — An object specifying protocol parameters to be used.

Exceptions

boost::system::system_error — Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
```

basic_socket::open (2 of 2 overloads)

Open the socket using the specified protocol.

```
boost::system::error_code open(
    const protocol_type & protocol,
    boost::system::error_code & ec);
```

This function opens the socket so that it will use the specified protocol.

Parameters

protocol — An object specifying which protocol is to be used.

ec — Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::system::error_code ec;
socket.open(boost::asio::ip::tcp::v4(), ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket::operator=

Move-assign a `basic_socket` from another.

```
basic_socket & operator=(
    basic_socket && other);
```

This assignment operator moves a socket from one object to another.

Parameters

other The other `basic_socket` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_socket(io_service&) constructor`.

basic_socket::protocol_type

The protocol type.

```
typedef Protocol protocol_type;
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::receive_buffer_size

Inherited from socket_base.

Socket option for the receive buffer size of a socket.

```
typedef implementation_defined receive_buffer_size;
```

Implements the SOL_SOCKET/SO_RCVBUF socket option.

Examples

Setting the option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::receive_low_watermark

Inherited from socket_base.

Socket option for the receive low watermark.

```
typedef implementation_defined receive_low_watermark;
```

Implements the SOL_SOCKET/SO_RCVLOWAT socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::remote_endpoint

Get the remote endpoint of the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
endpoint_type remote_endpoint() const;
  » more...

endpoint_type remote_endpoint(
    boost::system::error_code & ec) const;
  » more...
```

basic_socket::remote_endpoint (1 of 2 overloads)

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint() const;
```

This function is used to obtain the remote endpoint of the socket.

Return Value

An object that represents the remote endpoint of the socket.

Exceptions

boost::system::system_error	Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::endpoint endpoint = socket.remote_endpoint();
```

basic_socket::remote_endpoint (2 of 2 overloads)

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint(
    boost::system::error_code & ec) const;
```

This function is used to obtain the remote endpoint of the socket.

Parameters

ec	Set to indicate what error occurred, if any.

Return Value

An object that represents the remote endpoint of the socket. Returns a default-constructed endpoint object if an error occurred.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
boost::asio::ip::tcp::endpoint endpoint = socket.remote_endpoint(ec);
if (ec)
{
  // An error occurred.
}
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket::reuse_address

Inherited from socket_base.

Socket option to allow the socket to be bound to an address that is already in use.

```
typedef implementation_defined reuse_address;
```

Implements the SOL_SOCKET/SO_REUSEADDR socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option(true);
acceptor.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::send_buffer_size

Inherited from socket_base.

Socket option for the send buffer size of a socket.

```
typedef implementation_defined send_buffer_size;
```

Implements the SOL_SOCKET/SO_SNDBUF socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::send_low_watermark

Inherited from socket_base.

Socket option for the send low watermark.

```
typedef implementation_defined send_low_watermark;
```

Implements the SOL_SOCKET/SO_SNDLOWAT socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef SocketService service_type;
```

Requirements

Header: `boost/asio/basic_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_socket::set_option

Set an option on the socket.

```
template<
    typename SettableSocketOption>
void set_option(
    const SettableSocketOption & option);
  » more...

template<
    typename SettableSocketOption>
boost::system::error_code set_option(
    const SettableSocketOption & option,
    boost::system::error_code & ec);
  » more...
```

basic_socket::set_option (1 of 2 overloads)

Set an option on the socket.

```
template<
    typename SettableSocketOption>
void set_option(
    const SettableSocketOption & option);
```

This function is used to set an option on the socket.

Parameters

option The new option value to be set on the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Setting the IPPROTO_TCP/TCP_NODELAY option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::no_delay option(true);
socket.set_option(option);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket::set_option (2 of 2 overloads)

Set an option on the socket.

```
template<
    typename SettableSocketOption>
boost::system::error_code set_option(
    const SettableSocketOption & option,
    boost::system::error_code & ec);
```

This function is used to set an option on the socket.

Parameters

option The new option value to be set on the socket.

ec Set to indicate what error occurred, if any.

Example

Setting the IPPROTO_TCP/TCP_NODELAY option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::no_delay option(true);
boost::system::error_code ec;
socket.set_option(option, ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket::shutdown

Disable sends or receives on the socket.

```
void shutdown(
    shutdown_type what);
  » more...

boost::system::error_code shutdown(
    shutdown_type what,
    boost::system::error_code & ec);
  » more...
```

basic_socket::shutdown (1 of 2 overloads)

Disable sends or receives on the socket.

```
void shutdown(
    shutdown_type what);
```

This function is used to disable send operations, receive operations, or both.

Parameters

what Determines what types of operation will no longer be allowed.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Example

Shutting down the send side of the socket:

```
boost::asio::ip::tcp::socket socket(io_service);
...
socket.shutdown(boost::asio::ip::tcp::socket::shutdown_send);
```

basic_socket::shutdown (2 of 2 overloads)

Disable sends or receives on the socket.

```
boost::system::error_code shutdown(
    shutdown_type what,
    boost::system::error_code & ec);
```

This function is used to disable send operations, receive operations, or both.

Parameters

what	Determines what types of operation will no longer be allowed.
ec	Set to indicate what error occurred, if any.

Example

Shutting down the send side of the socket:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
socket.shutdown(boost::asio::ip::tcp::socket::shutdown_send, ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket::shutdown_type

Inherited from socket_base.

Different ways a socket may be shutdown.

```
enum shutdown_type
```

Values

shutdown_receive	Shutdown the receive side of the socket.
shutdown_send	Shutdown the send side of the socket.
shutdown_both	Shutdown both send and receive on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket::~basic_socket

Protected destructor to prevent deletion through this type.

```
~basic_socket();
```

basic_socket_acceptor

Provides the ability to accept new connections.

```
template<
    typename Protocol,
    typename SocketAcceptorService = socket_acceptor_service<Protocol>>
class basic_socket_acceptor :
  public basic_io_object< SocketAcceptorService >,
  public socket_base
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of an acceptor.
native_type	(Deprecated: Use native_handle_type.) The native representation of an acceptor.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
accept	Accept a new connection. Accept a new connection and obtain the endpoint of the peer.
assign	Assigns an existing native acceptor to the acceptor.
async_accept	Start an asynchronous accept.
basic_socket_acceptor	Construct an acceptor without opening it. Construct an open acceptor. Construct an acceptor opened on the given endpoint. Construct a basic_socket_acceptor on an existing native acceptor. Move-construct a basic_socket_acceptor from another.
bind	Bind the acceptor to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the acceptor.
close	Close the acceptor.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the acceptor.
io_control	Perform an IO control command on the acceptor.
is_open	Determine whether the acceptor is open.
listen	Place the acceptor into the state where it will listen for new connections.
local_endpoint	Get the local endpoint of the acceptor.
native	(Deprecated: Use native_handle().) Get the native acceptor representation.
native_handle	Get the native acceptor representation.
native_non_blocking	Gets the non-blocking mode of the native acceptor implementation. Sets the non-blocking mode of the native acceptor implementation.
non_blocking	Gets the non-blocking mode of the acceptor. Sets the non-blocking mode of the acceptor.
open	Open the acceptor using the specified protocol.
operator=	Move-assign a basic_socket_acceptor from another.
set_option	Set an option on the acceptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_socket_acceptor` class template is used for accepting new socket connections.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Example

Opening a socket acceptor with the SO_REUSEADDR option enabled:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
boost::asio::ip::tcp::endpoint endpoint(boost::asio::ip::tcp::v4(), port);
acceptor.open(endpoint.protocol());
acceptor.set_option(boost::asio::ip::tcp::acceptor::reuse_address(true));
acceptor.bind(endpoint);
acceptor.listen();
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::accept

Accept a new connection.

```
template<
    typename SocketService>
void accept(
    basic_socket< protocol_type, SocketService > & peer);
  » more...

template<
    typename SocketService>
boost::system::error_code accept(
    basic_socket< protocol_type, SocketService > & peer,
    boost::system::error_code & ec);
  » more...
```

Accept a new connection and obtain the endpoint of the peer.

```
template<
    typename SocketService>
void accept(
    basic_socket< protocol_type, SocketService > & peer,
    endpoint_type & peer_endpoint);
  » more...

template<
    typename SocketService>
boost::system::error_code accept(
    basic_socket< protocol_type, SocketService > & peer,
    endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
  » more...
```

basic_socket_acceptor::accept (1 of 4 overloads)

Accept a new connection.

```
template<
    typename SocketService>
void accept(
    basic_socket< protocol_type, SocketService > & peer);
```

This function is used to accept a new connection from a peer into the given socket. The function call will block until a new connection has been accepted successfully or an error occurs.

Parameters

peer The socket into which the new connection will be accepted.

Exceptions

boost::system::system_error Thrown on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::ip::tcp::socket socket(io_service);
acceptor.accept(socket);
```

basic_socket_acceptor::accept (2 of 4 overloads)

Accept a new connection.

```
template<
    typename SocketService>
boost::system::error_code accept(
    basic_socket< protocol_type, SocketService > & peer,
    boost::system::error_code & ec);
```

This function is used to accept a new connection from a peer into the given socket. The function call will block until a new connection has been accepted successfully or an error occurs.

Parameters

peer The socket into which the new connection will be accepted.

ec Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::ip::tcp::soocket socket(io_service);
boost::system::error_code ec;
acceptor.accept(socket, ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_acceptor::accept (3 of 4 overloads)

Accept a new connection and obtain the endpoint of the peer.

```
template<
    typename SocketService>
void accept(
    basic_socket< protocol_type, SocketService > & peer,
    endpoint_type & peer_endpoint);
```

This function is used to accept a new connection from a peer into the given socket, and additionally provide the endpoint of the remote peer. The function call will block until a new connection has been accepted successfully or an error occurs.

Parameters

peer The socket into which the new connection will be accepted.

peer_endpoint An endpoint object which will receive the endpoint of the remote peer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Example

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint;
acceptor.accept(socket, endpoint);
```

basic_socket_acceptor::accept (4 of 4 overloads)

Accept a new connection and obtain the endpoint of the peer.

```
template<
    typename SocketService>
boost::system::error_code accept(
    basic_socket< protocol_type, SocketService > & peer,
    endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
```

This function is used to accept a new connection from a peer into the given socket, and additionally provide the endpoint of the remote peer. The function call will block until a new connection has been accepted successfully or an error occurs.

Parameters

peer	The socket into which the new connection will be accepted.
peer_endpoint	An endpoint object which will receive the endpoint of the remote peer.
ec	Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint;
boost::system::error_code ec;
acceptor.accept(socket, endpoint, ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_acceptor::assign

Assigns an existing native acceptor to the acceptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void assign(
    const protocol_type & protocol,
    const native_handle_type & native_acceptor);
  » more...

boost::system::error_code assign(
    const protocol_type & protocol,
    const native_handle_type & native_acceptor,
    boost::system::error_code & ec);
  » more...
```

basic_socket_acceptor::assign (1 of 2 overloads)

Assigns an existing native acceptor to the acceptor.

```
void assign(
    const protocol_type & protocol,
    const native_handle_type & native_acceptor);
```

basic_socket_acceptor::assign (2 of 2 overloads)

Assigns an existing native acceptor to the acceptor.

```
boost::system::error_code assign(
    const protocol_type & protocol,
    const native_handle_type & native_acceptor,
    boost::system::error_code & ec);
```

basic_socket_acceptor::async_accept

Start an asynchronous accept.

```
template<
    typename SocketService,
    typename AcceptHandler>
void async_accept(
    basic_socket< protocol_type, SocketService > & peer,
    AcceptHandler handler);
  » more...

template<
    typename SocketService,
    typename AcceptHandler>
void async_accept(
    basic_socket< protocol_type, SocketService > & peer,
    endpoint_type & peer_endpoint,
    AcceptHandler handler);
  » more...
```

basic_socket_acceptor::async_accept (1 of 2 overloads)

Start an asynchronous accept.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename SocketService,
    typename AcceptHandler>
void async_accept(
    basic_socket< protocol_type, SocketService > & peer,
    AcceptHandler handler);
```

This function is used to asynchronously accept a new connection into a socket. The function call always returns immediately.

Parameters

peer — The socket into which the new connection will be accepted. Ownership of the peer object is retained by the caller, which must guarantee that it is valid until the handler is called.

handler — The handler to be called when the accept operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error // Result of operation.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

```
void accept_handler(const boost::system::error_code& error)
{
  if (!error)
  {
    // Accept succeeded.
  }
}

...

boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::ip::tcp::socket socket(io_service);
acceptor.async_accept(socket, accept_handler);
```

basic_socket_acceptor::async_accept (2 of 2 overloads)

Start an asynchronous accept.

```
template<
    typename SocketService,
    typename AcceptHandler>
void async_accept(
    basic_socket< protocol_type, SocketService > & peer,
    endpoint_type & peer_endpoint,
    AcceptHandler handler);
```

This function is used to asynchronously accept a new connection into a socket, and additionally obtain the endpoint of the remote peer. The function call always returns immediately.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

peer	The socket into which the new connection will be accepted. Ownership of the peer object is retained by the caller, which must guarantee that it is valid until the handler is called.
peer_endpoint	An endpoint object into which the endpoint of the remote peer will be written. Ownership of the peer_endpoint object is retained by the caller, which must guarantee that it is valid until the handler is called.
handler	The handler to be called when the accept operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error // Result of operation.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

basic_socket_acceptor::basic_socket_acceptor

Construct an acceptor without opening it.

```
explicit basic_socket_acceptor(
    boost::asio::io_service & io_service);
  » more...
```

Construct an open acceptor.

```
basic_socket_acceptor(
    boost::asio::io_service & io_service,
    const protocol_type & protocol);
  » more...
```

Construct an acceptor opened on the given endpoint.

```
basic_socket_acceptor(
    boost::asio::io_service & io_service,
    const endpoint_type & endpoint,
    bool reuse_addr = true);
  » more...
```

Construct a `basic_socket_acceptor` on an existing native acceptor.

```
basic_socket_acceptor(
    boost::asio::io_service & io_service,
    const protocol_type & protocol,
    const native_handle_type & native_acceptor);
  » more...
```

Move-construct a `basic_socket_acceptor` from another.

```
basic_socket_acceptor(
    basic_socket_acceptor && other);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_acceptor::basic_socket_acceptor (1 of 5 overloads)

Construct an acceptor without opening it.

```
basic_socket_acceptor(
    boost::asio::io_service & io_service);
```

This constructor creates an acceptor without opening it to listen for new connections. The `open()` function must be called before the acceptor can accept new socket connections.

Parameters

io_service	The `io_service` object that the acceptor will use to dispatch handlers for any asynchronous operations performed on the acceptor.

basic_socket_acceptor::basic_socket_acceptor (2 of 5 overloads)

Construct an open acceptor.

```
basic_socket_acceptor(
    boost::asio::io_service & io_service,
    const protocol_type & protocol);
```

This constructor creates an acceptor and automatically opens it.

Parameters

io_service	The `io_service` object that the acceptor will use to dispatch handlers for any asynchronous operations performed on the acceptor.
protocol	An object specifying protocol parameters to be used.

Exceptions

boost::system::system_error	Thrown on failure.

basic_socket_acceptor::basic_socket_acceptor (3 of 5 overloads)

Construct an acceptor opened on the given endpoint.

```
basic_socket_acceptor(
    boost::asio::io_service & io_service,
    const endpoint_type & endpoint,
    bool reuse_addr = true);
```

This constructor creates an acceptor and automatically opens it to listen for new connections on the specified endpoint.

Parameters

io_service	The `io_service` object that the acceptor will use to dispatch handlers for any asynchronous operations performed on the acceptor.
endpoint	An endpoint on the local machine on which the acceptor will listen for new connections.
reuse_addr	Whether the constructor should set the socket option `socket_base::reuse_address`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

This constructor is equivalent to the following code:

```
basic_socket_acceptor<Protocol> acceptor(io_service);
acceptor.open(endpoint.protocol());
if (reuse_addr)
  acceptor.set_option(socket_base::reuse_address(true));
acceptor.bind(endpoint);
acceptor.listen(listen_backlog);
```

basic_socket_acceptor::basic_socket_acceptor (4 of 5 overloads)

Construct a `basic_socket_acceptor` on an existing native acceptor.

```
basic_socket_acceptor(
    boost::asio::io_service & io_service,
    const protocol_type & protocol,
    const native_handle_type & native_acceptor);
```

This constructor creates an acceptor object to hold an existing native acceptor.

Parameters

io_service	The `io_service` object that the acceptor will use to dispatch handlers for any asynchronous operations performed on the acceptor.
protocol	An object specifying protocol parameters to be used.
native_acceptor	A native acceptor.

Exceptions

boost::system::system_error	Thrown on failure.

basic_socket_acceptor::basic_socket_acceptor (5 of 5 overloads)

Move-construct a `basic_socket_acceptor` from another.

```
basic_socket_acceptor(
    basic_socket_acceptor && other);
```

This constructor moves an acceptor from one object to another.

Parameters

other	The other `basic_socket_acceptor` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_socket_acceptor(io_service&)` constructor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_acceptor::bind

Bind the acceptor to the given local endpoint.

```
void bind(
    const endpoint_type & endpoint);
  » more...

boost::system::error_code bind(
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
  » more...
```

basic_socket_acceptor::bind (1 of 2 overloads)

Bind the acceptor to the given local endpoint.

```
void bind(
    const endpoint_type & endpoint);
```

This function binds the socket acceptor to the specified endpoint on the local machine.

Parameters

endpoint An endpoint on the local machine to which the socket acceptor will be bound.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
acceptor.open(boost::asio::ip::tcp::v4());
acceptor.bind(boost::asio::ip::tcp::endpoint(12345));
```

basic_socket_acceptor::bind (2 of 2 overloads)

Bind the acceptor to the given local endpoint.

```
boost::system::error_code bind(
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
```

This function binds the socket acceptor to the specified endpoint on the local machine.

Parameters

endpoint An endpoint on the local machine to which the socket acceptor will be bound.

ec Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
acceptor.open(boost::asio::ip::tcp::v4());
boost::system::error_code ec;
acceptor.bind(boost::asio::ip::tcp::endpoint(12345), ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_acceptor::broadcast

Inherited from socket_base.

Socket option to permit sending of broadcast messages.

```
typedef implementation_defined broadcast;
```

Implements the SOL_SOCKET/SO_BROADCAST socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::bytes_readable

Inherited from socket_base.

IO control command to get the amount of data that can be read without blocking.

```
typedef implementation_defined bytes_readable;
```

Implements the FIONREAD IO control command.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::bytes_readable command(true);
socket.io_control(command);
std::size_t bytes_readable = command.get();
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::cancel

Cancel all asynchronous operations associated with the acceptor.

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

basic_socket_acceptor::cancel (1 of 2 overloads)

Cancel all asynchronous operations associated with the acceptor.

```
void cancel();
```

This function causes all outstanding asynchronous connect, send and receive operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure.

basic_socket_acceptor::cancel (2 of 2 overloads)

Cancel all asynchronous operations associated with the acceptor.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function causes all outstanding asynchronous connect, send and receive operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Parameters

ec	Set to indicate what error occurred, if any.

basic_socket_acceptor::close

Close the acceptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

basic_socket_acceptor::close (1 of 2 overloads)

Close the acceptor.

```
void close();
```

This function is used to close the acceptor. Any asynchronous accept operations will be cancelled immediately.

A subsequent call to `open()` is required before the acceptor can again be used to again perform socket accept operations.

Exceptions

boost::system::system_error Thrown on failure.

basic_socket_acceptor::close (2 of 2 overloads)

Close the acceptor.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```

This function is used to close the acceptor. Any asynchronous accept operations will be cancelled immediately.

A subsequent call to `open()` is required before the acceptor can again be used to again perform socket accept operations.

Parameters

ec Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::system::error_code ec;
acceptor.close(ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_acceptor::debug

Inherited from socket_base.

Socket option to enable socket-level debugging.

```
typedef implementation_defined debug;
```

Implements the SOL_SOCKET/SO_DEBUG socket option.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::do_not_route

Inherited from socket_base.

Socket option to prevent routing, use local interfaces only.

```
typedef implementation_defined do_not_route;
```

Implements the SOL_SOCKET/SO_DONTROUTE socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_acceptor::enable_connection_aborted

Inherited from socket_base.

Socket option to report aborted connections on accept.

```
typedef implementation_defined enable_connection_aborted;
```

Implements a custom socket option that determines whether or not an accept operation is permitted to fail with `boost::asio::error::connection_aborted`. By default the option is false.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option(true);
acceptor.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::endpoint_type

The endpoint type.

```
typedef Protocol::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_acceptor::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

basic_socket_acceptor::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

basic_socket_acceptor::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

basic_socket_acceptor::get_option

Get an option from the acceptor.

```
template<
    typename GettableSocketOption>
void get_option(
    GettableSocketOption & option);
  » more...

template<
    typename GettableSocketOption>
boost::system::error_code get_option(
    GettableSocketOption & option,
    boost::system::error_code & ec);
  » more...
```

basic_socket_acceptor::get_option (1 of 2 overloads)

Get an option from the acceptor.

```
template<
    typename GettableSocketOption>
void get_option(
    GettableSocketOption & option);
```

This function is used to get the current value of an option on the acceptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

option The option value to be obtained from the acceptor.

Exceptions

boost::system::system_error Thrown on failure.

Example

Getting the value of the SOL_SOCKET/SO_REUSEADDR option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::ip::tcp::acceptor::reuse_address option;
acceptor.get_option(option);
bool is_set = option.get();
```

basic_socket_acceptor::get_option (2 of 2 overloads)

Get an option from the acceptor.

```
template<
    typename GettableSocketOption>
boost::system::error_code get_option(
    GettableSocketOption & option,
    boost::system::error_code & ec);
```

This function is used to get the current value of an option on the acceptor.

Parameters

option The option value to be obtained from the acceptor.

ec Set to indicate what error occurred, if any.

Example

Getting the value of the SOL_SOCKET/SO_REUSEADDR option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::ip::tcp::acceptor::reuse_address option;
boost::system::error_code ec;
acceptor.get_option(option, ec);
if (ec)
{
  // An error occurred.
}
bool is_set = option.get();
```

basic_socket_acceptor::get_service

Get the service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

basic_socket_acceptor::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

basic_socket_acceptor::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

basic_socket_acceptor::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

basic_socket_acceptor::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::io_control

Perform an IO control command on the acceptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename IoControlCommand>
void io_control(
    IoControlCommand & command);
  » more...

template<
    typename IoControlCommand>
boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
  » more...
```

basic_socket_acceptor::io_control (1 of 2 overloads)

Perform an IO control command on the acceptor.

```
template<
    typename IoControlCommand>
void io_control(
    IoControlCommand & command);
```

This function is used to execute an IO control command on the acceptor.

Parameters

command The IO control command to be performed on the acceptor.

Exceptions

boost::system::system_error Thrown on failure.

Example

Getting the number of bytes ready to read:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::ip::tcp::acceptor::non_blocking_io command(true);
socket.io_control(command);
```

basic_socket_acceptor::io_control (2 of 2 overloads)

Perform an IO control command on the acceptor.

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
```

This function is used to execute an IO control command on the acceptor.

Parameters

command The IO control command to be performed on the acceptor.

ec Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

Getting the number of bytes ready to read:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::ip::tcp::acceptor::non_blocking_io command(true);
boost::system::error_code ec;
socket.io_control(command, ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_acceptor::is_open

Determine whether the acceptor is open.

```
bool is_open() const;
```

basic_socket_acceptor::keep_alive

Inherited from socket_base.

Socket option to send keep-alives.

```
typedef implementation_defined keep_alive;
```

Implements the SOL_SOCKET/SO_KEEPALIVE socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::linger

Inherited from socket_base.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Socket option to specify whether the socket lingers on close if unsent data is present.

```
typedef implementation_defined linger;
```

Implements the SOL_SOCKET/SO_LINGER socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option(true, 30);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option;
socket.get_option(option);
bool is_set = option.enabled();
unsigned short timeout = option.timeout();
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::listen

Place the acceptor into the state where it will listen for new connections.

```
void listen(
    int backlog = socket_base::max_connections);
  » more...

boost::system::error_code listen(
    int backlog,
    boost::system::error_code & ec);
  » more...
```

basic_socket_acceptor::listen (1 of 2 overloads)

Place the acceptor into the state where it will listen for new connections.

```
void listen(
    int backlog = socket_base::max_connections);
```

This function puts the socket acceptor into the state where it may accept new connections.

Parameters

backlog The maximum length of the queue of pending connections.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

basic_socket_acceptor::listen (2 of 2 overloads)

Place the acceptor into the state where it will listen for new connections.

```
boost::system::error_code listen(
    int backlog,
    boost::system::error_code & ec);
```

This function puts the socket acceptor into the state where it may accept new connections.

Parameters

backlog	The maximum length of the queue of pending connections.
ec	Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::system::error_code ec;
acceptor.listen(boost::asio::socket_base::max_connections, ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_acceptor::local_endpoint

Get the local endpoint of the acceptor.

```
endpoint_type local_endpoint() const;
  » more...

endpoint_type local_endpoint(
    boost::system::error_code & ec) const;
  » more...
```

basic_socket_acceptor::local_endpoint (1 of 2 overloads)

Get the local endpoint of the acceptor.

```
endpoint_type local_endpoint() const;
```

This function is used to obtain the locally bound endpoint of the acceptor.

Return Value

An object that represents the local endpoint of the acceptor.

Exceptions

boost::system::system_error	Thrown on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::ip::tcp::endpoint endpoint = acceptor.local_endpoint();
```

basic_socket_acceptor::local_endpoint (2 of 2 overloads)

Get the local endpoint of the acceptor.

```
endpoint_type local_endpoint(
    boost::system::error_code & ec) const;
```

This function is used to obtain the locally bound endpoint of the acceptor.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

An object that represents the local endpoint of the acceptor. Returns a default-constructed endpoint object if an error occurred and the error handler did not throw an exception.

Example

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::system::error_code ec;
boost::asio::ip::tcp::endpoint endpoint = acceptor.local_endpoint(ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_acceptor::max_connections

Inherited from socket_base.

The maximum length of the queue of pending incoming connections.

```
static const int max_connections = implementation_defined;
```

basic_socket_acceptor::message_do_not_route

Inherited from socket_base.

Specify that the data should not be subject to routing.

```
static const int message_do_not_route = implementation_defined;
```

basic_socket_acceptor::message_end_of_record

Inherited from socket_base.

Specifies that the data marks the end of a record.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
static const int message_end_of_record = implementation_defined;
```

basic_socket_acceptor::message_flags

Inherited from socket_base.

Bitmask type for flags that can be passed to send and receive operations.

```
typedef int message_flags;
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::message_out_of_band

Inherited from socket_base.

Process out-of-band data.

```
static const int message_out_of_band = implementation_defined;
```

basic_socket_acceptor::message_peek

Inherited from socket_base.

Peek at incoming data without removing it from the input queue.

```
static const int message_peek = implementation_defined;
```

basic_socket_acceptor::native

(Deprecated: Use `native_handle()`.) Get the native acceptor representation.

```
native_type native();
```

This function may be used to obtain the underlying representation of the acceptor. This is intended to allow access to native acceptor functionality that is not otherwise provided.

basic_socket_acceptor::native_handle

Get the native acceptor representation.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying representation of the acceptor. This is intended to allow access to native acceptor functionality that is not otherwise provided.

basic_socket_acceptor::native_handle_type

The native representation of an acceptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef SocketAcceptorService::native_handle_type native_handle_type;
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::native_non_blocking

Gets the non-blocking mode of the native acceptor implementation.

```
bool native_non_blocking() const;
  » more...
```

Sets the non-blocking mode of the native acceptor implementation.

```
void native_non_blocking(
    bool mode);
  » more...

boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

basic_socket_acceptor::native_non_blocking (1 of 3 overloads)

Gets the non-blocking mode of the native acceptor implementation.

```
bool native_non_blocking() const;
```

This function is used to retrieve the non-blocking mode of the underlying native acceptor. This mode has no effect on the behaviour of the acceptor object's synchronous operations.

Return Value

`true` if the underlying acceptor is in non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Remarks

The current non-blocking mode is cached by the acceptor object. Consequently, the return value may be incorrect if the non-blocking mode was set directly on the native acceptor.

basic_socket_acceptor::native_non_blocking (2 of 3 overloads)

Sets the non-blocking mode of the native acceptor implementation.

```
void native_non_blocking(
    bool mode);
```

This function is used to modify the non-blocking mode of the underlying native acceptor. It has no effect on the behaviour of the acceptor object's synchronous operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

mode — If `true`, the underlying acceptor is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Exceptions

boost::system::system_error — Thrown on failure. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

basic_socket_acceptor::native_non_blocking (3 of 3 overloads)

Sets the non-blocking mode of the native acceptor implementation.

```
boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

This function is used to modify the non-blocking mode of the underlying native acceptor. It has no effect on the behaviour of the acceptor object's synchronous operations.

Parameters

mode — If `true`, the underlying acceptor is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

ec — Set to indicate what error occurred, if any. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

basic_socket_acceptor::native_type

(Deprecated: Use native_handle_type.) The native representation of an acceptor.

```
typedef SocketAcceptorService::native_handle_type native_type;
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::non_blocking

Gets the non-blocking mode of the acceptor.

```
bool non_blocking() const;
  » more...
```

Sets the non-blocking mode of the acceptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void non_blocking(
    bool mode);
  » more...

boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

basic_socket_acceptor::non_blocking (1 of 3 overloads)

Gets the non-blocking mode of the acceptor.

```
bool non_blocking() const;
```

Return Value

`true` if the acceptor's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_socket_acceptor::non_blocking (2 of 3 overloads)

Sets the non-blocking mode of the acceptor.

```
void non_blocking(
    bool mode);
```

Parameters

mode — If `true`, the acceptor's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Exceptions

boost::system::system_error — Thrown on failure.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_socket_acceptor::non_blocking (3 of 3 overloads)

Sets the non-blocking mode of the acceptor.

```
boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

Parameters

mode — If `true`, the acceptor's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ec Set to indicate what error occurred, if any.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_socket_acceptor::non_blocking_io

Inherited from socket_base.

(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.

```
typedef implementation_defined non_blocking_io;
```

Implements the FIONBIO IO control command.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::non_blocking_io command(true);
socket.io_control(command);
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::open

Open the acceptor using the specified protocol.

```
void open(
    const protocol_type & protocol = protocol_type());
  » more...

boost::system::error_code open(
    const protocol_type & protocol,
    boost::system::error_code & ec);
  » more...
```

basic_socket_acceptor::open (1 of 2 overloads)

Open the acceptor using the specified protocol.

```
void open(
    const protocol_type & protocol = protocol_type());
```

This function opens the socket acceptor so that it will use the specified protocol.

Parameters

protocol An object specifying which protocol is to be used.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Example

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
acceptor.open(boost::asio::ip::tcp::v4());
```

basic_socket_acceptor::open (2 of 2 overloads)

Open the acceptor using the specified protocol.

```
boost::system::error_code open(
    const protocol_type & protocol,
    boost::system::error_code & ec);
```

This function opens the socket acceptor so that it will use the specified protocol.

Parameters

protocol	An object specifying which protocol is to be used.
ec	Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
boost::system::error_code ec;
acceptor.open(boost::asio::ip::tcp::v4(), ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_acceptor::operator=

Move-assign a `basic_socket_acceptor` from another.

```
basic_socket_acceptor & operator=(
    basic_socket_acceptor && other);
```

This assignment operator moves an acceptor from one object to another.

Parameters

other	The other `basic_socket_acceptor` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_socket_acceptor(io_service&)` constructor.

basic_socket_acceptor::protocol_type

The protocol type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef Protocol protocol_type;
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::receive_buffer_size

Inherited from socket_base.

Socket option for the receive buffer size of a socket.

```
typedef implementation_defined receive_buffer_size;
```

Implements the SOL_SOCKET/SO_RCVBUF socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::receive_low_watermark

Inherited from socket_base.

Socket option for the receive low watermark.

```
typedef implementation_defined receive_low_watermark;
```

Implements the SOL_SOCKET/SO_RCVLOWAT socket option.

Examples

Setting the option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::reuse_address

Inherited from socket_base.

Socket option to allow the socket to be bound to an address that is already in use.

```
typedef implementation_defined reuse_address;
```

Implements the SOL_SOCKET/SO_REUSEADDR socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option(true);
acceptor.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::send_buffer_size

Inherited from socket_base.

Socket option for the send buffer size of a socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef implementation_defined send_buffer_size;
```

Implements the SOL_SOCKET/SO_SNDBUF socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::send_low_watermark

Inherited from socket_base.

Socket option for the send low watermark.

```
typedef implementation_defined send_low_watermark;
```

Implements the SOL_SOCKET/SO_SNDLOWAT socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

basic_socket_acceptor::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef SocketAcceptorService service_type;
```

Requirements

Header: `boost/asio/basic_socket_acceptor.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_acceptor::set_option

Set an option on the acceptor.

```
template<
    typename SettableSocketOption>
void set_option(
    const SettableSocketOption & option);
  » more...

template<
    typename SettableSocketOption>
boost::system::error_code set_option(
    const SettableSocketOption & option,
    boost::system::error_code & ec);
  » more...
```

basic_socket_acceptor::set_option (1 of 2 overloads)

Set an option on the acceptor.

```
template<
    typename SettableSocketOption>
void set_option(
    const SettableSocketOption & option);
```

This function is used to set an option on the acceptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

option The new option value to be set on the acceptor.

Exceptions

boost::system::system_error Thrown on failure.

Example

Setting the SOL_SOCKET/SO_REUSEADDR option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::ip::tcp::acceptor::reuse_address option(true);
acceptor.set_option(option);
```

basic_socket_acceptor::set_option (2 of 2 overloads)

Set an option on the acceptor.

```
template<
    typename SettableSocketOption>
boost::system::error_code set_option(
    const SettableSocketOption & option,
    boost::system::error_code & ec);
```

This function is used to set an option on the acceptor.

Parameters

option The new option value to be set on the acceptor.

ec Set to indicate what error occurred, if any.

Example

Setting the SOL_SOCKET/SO_REUSEADDR option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::ip::tcp::acceptor::reuse_address option(true);
boost::system::error_code ec;
acceptor.set_option(option, ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_acceptor::shutdown_type

Inherited from socket_base.

Different ways a socket may be shutdown.

```
enum shutdown_type
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Values

shutdown_receive	Shutdown the receive side of the socket.
shutdown_send	Shutdown the send side of the socket.
shutdown_both	Shutdown both send and receive on the socket.

basic_socket_iostream

Iostream interface for a socket.

```
template<
    typename Protocol,
    typename StreamSocketService = stream_socket_service<Protocol>,
    typename Time = boost::posix_time::ptime,
    typename TimeTraits = boost::asio::time_traits<Time>,
    typename TimerService = deadline_timer_service<Time, TimeTraits>>
class basic_socket_iostream
```

Types

Name	Description
duration_type	The duration type.
endpoint_type	The endpoint type.
time_type	The time type.

Member Functions

Name	Description
basic_socket_iostream	Construct a basic_socket_iostream without establishing a connection. Establish a connection to an endpoint corresponding to a resolver query.
close	Close the connection.
connect	Establish a connection to an endpoint corresponding to a resolver query.
error	Get the last error associated with the stream.
expires_at	Get the stream's expiry time as an absolute time. Set the stream's expiry time as an absolute time.
expires_from_now	Get the timer's expiry time relative to now. Set the stream's expiry time relative to now.
rdbuf	Return a pointer to the underlying streambuf.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/basic_socket_iostream.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_iostream::basic_socket_iostream

Construct a `basic_socket_iostream` without establishing a connection.

```
basic_socket_iostream();
  » more...
```

Establish a connection to an endpoint corresponding to a resolver query.

```
template<
    typename T1,
    ... ,
    typename TN>
explicit basic_socket_iostream(
    T1 t1,
    ... ,
    TN tn);
  » more...
```

basic_socket_iostream::basic_socket_iostream (1 of 2 overloads)

Construct a `basic_socket_iostream` without establishing a connection.

```
basic_socket_iostream();
```

basic_socket_iostream::basic_socket_iostream (2 of 2 overloads)

Establish a connection to an endpoint corresponding to a resolver query.

```
template<
    typename T1,
    ... ,
    typename TN>
basic_socket_iostream(
    T1 t1,
    ... ,
    TN tn);
```

This constructor automatically establishes a connection based on the supplied resolver query parameters. The arguments are used to construct a resolver query object.

basic_socket_iostream::close

Close the connection.

```
void close();
```

basic_socket_iostream::connect

Establish a connection to an endpoint corresponding to a resolver query.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename T1,
    ... ,
    typename TN>
void connect(
    T1 t1,
    ... ,
    TN tn);
```

This function automatically establishes a connection based on the supplied resolver query parameters. The arguments are used to construct a resolver query object.

basic_socket_iostream::duration_type

The duration type.

```
typedef TimeTraits::duration_type duration_type;
```

Requirements

Header: `boost/asio/basic_socket_iostream.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_iostream::endpoint_type

The endpoint type.

```
typedef Protocol::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/basic_socket_iostream.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_iostream::error

Get the last error associated with the stream.

```
const boost::system::error_code & error() const;
```

Return Value

An `error_code` corresponding to the last error from the stream.

Example

To print the error associated with a failure to establish a connection:

```
tcp::iostream s("www.boost.org", "http");
if (!s)
{
  std::cout << "Error: " << s.error().message() << std::endl;
}
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_iostream::expires_at

Get the stream's expiry time as an absolute time.

```
time_type expires_at() const;
  » more...
```

Set the stream's expiry time as an absolute time.

```
void expires_at(
    const time_type & expiry_time);
  » more...
```

basic_socket_iostream::expires_at (1 of 2 overloads)

Get the stream's expiry time as an absolute time.

```
time_type expires_at() const;
```

Return Value

An absolute time value representing the stream's expiry time.

basic_socket_iostream::expires_at (2 of 2 overloads)

Set the stream's expiry time as an absolute time.

```
void expires_at(
    const time_type & expiry_time);
```

This function sets the expiry time associated with the stream. Stream operations performed after this time (where the operations cannot be completed using the internal buffers) will fail with the error `boost::asio::error::operation_aborted`.

Parameters

expiry_time	The expiry time to be used for the stream.

basic_socket_iostream::expires_from_now

Get the timer's expiry time relative to now.

```
duration_type expires_from_now() const;
  » more...
```

Set the stream's expiry time relative to now.

```
void expires_from_now(
    const duration_type & expiry_time);
  » more...
```

basic_socket_iostream::expires_from_now (1 of 2 overloads)

Get the timer's expiry time relative to now.

```
duration_type expires_from_now() const;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Return Value

A relative time value representing the stream's expiry time.

basic_socket_iostream::expires_from_now (2 of 2 overloads)

Set the stream's expiry time relative to now.

```
void expires_from_now(
    const duration_type & expiry_time);
```

This function sets the expiry time associated with the stream. Stream operations performed after this time (where the operations cannot be completed using the internal buffers) will fail with the error `boost::asio::error::operation_aborted`.

Parameters

expiry_time The expiry time to be used for the timer.

basic_socket_iostream::rdbuf

Return a pointer to the underlying streambuf.

```
basic_socket_streambuf< Protocol, StreamSocketService, Time, TimeTraits, TimerService > * rd↵
buf() const;
```

basic_socket_iostream::time_type

The time type.

```
typedef TimeTraits::time_type time_type;
```

Requirements

Header: `boost/asio/basic_socket_iostream.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf

Iostream streambuf for a socket.

```
template<
    typename Protocol,
    typename StreamSocketService = stream_socket_service<Protocol>,
    typename Time = boost::posix_time::ptime,
    typename TimeTraits = boost::asio::time_traits<Time>,
    typename TimerService = deadline_timer_service<Time, TimeTraits>>
class basic_socket_streambuf :
  public basic_socket< Protocol, StreamSocketService >
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
duration_type	The duration type.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.
time_type	The time type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_socket_streambuf	Construct a basic_socket_streambuf without establishing a connection.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the connection. Close the socket.
connect	Establish a connection. Connect the socket to the specified endpoint.
expires_at	Get the stream buffer's expiry time as an absolute time. Set the stream buffer's expiry time as an absolute time.
expires_from_now	Get the stream buffer's expiry time relative to now. Set the stream buffer's expiry time relative to now.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
open	Open the socket using the specified protocol.
puberror	Get the last error associated with the stream buffer.
remote_endpoint	Get the remote endpoint of the socket.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.
~basic_socket_streambuf	Destructor flushes buffered data.

Protected Member Functions

Name	Description
error	Get the last error associated with the stream buffer.
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
overflow	
setbuf	
sync	
underflow	

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

Friends

Name	Description
io_handler	
timer_handler	

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::assign

Assign an existing native socket to the socket.

```
void assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket);
  » more...

boost::system::error_code assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
  » more...
```

basic_socket_streambuf::assign (1 of 2 overloads)

Inherited from basic_socket.

Assign an existing native socket to the socket.

```
void assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket);
```

basic_socket_streambuf::assign (2 of 2 overloads)

Inherited from basic_socket.

Assign an existing native socket to the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
```

basic_socket_streambuf::async_connect

Inherited from basic_socket.

Start an asynchronous connect.

```
template<
    typename ConnectHandler>
void async_connect(
    const endpoint_type & peer_endpoint,
    ConnectHandler handler);
```

This function is used to asynchronously connect a socket to the specified remote endpoint. The function call always returns immediately.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint — The remote endpoint to which the socket will be connected. Copies will be made of the endpoint object as required.

handler — The handler to be called when the connection operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error // Result of operation
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

```
void connect_handler(const boost::system::error_code& error)
{
  if (!error)
  {
    // Connect succeeded.
  }
}

...

boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.async_connect(endpoint, connect_handler);
```

basic_socket_streambuf::at_mark

Determine whether the socket is at the out-of-band data mark.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
bool at_mark() const;
  » more...

bool at_mark(
    boost::system::error_code & ec) const;
  » more...
```

basic_socket_streambuf::at_mark (1 of 2 overloads)

Inherited from basic_socket.

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark() const;
```

This function is used to check whether the socket input is currently positioned at the out-of-band data mark.

Return Value

A bool indicating whether the socket is at the out-of-band data mark.

Exceptions

boost::system::system_error Thrown on failure.

basic_socket_streambuf::at_mark (2 of 2 overloads)

Inherited from basic_socket.

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark(
    boost::system::error_code & ec) const;
```

This function is used to check whether the socket input is currently positioned at the out-of-band data mark.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

A bool indicating whether the socket is at the out-of-band data mark.

basic_socket_streambuf::available

Determine the number of bytes available for reading.

```
std::size_t available() const;
  » more...

std::size_t available(
    boost::system::error_code & ec) const;
  » more...
```

basic_socket_streambuf::available (1 of 2 overloads)

Inherited from basic_socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Determine the number of bytes available for reading.

```
std::size_t available() const;
```

This function is used to determine the number of bytes that may be read without blocking.

Return Value

The number of bytes that may be read without blocking, or 0 if an error occurs.

Exceptions

boost::system::system_error Thrown on failure.

basic_socket_streambuf::available (2 of 2 overloads)

Inherited from basic_socket.

Determine the number of bytes available for reading.

```
std::size_t available(
    boost::system::error_code & ec) const;
```

This function is used to determine the number of bytes that may be read without blocking.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes that may be read without blocking, or 0 if an error occurs.

basic_socket_streambuf::basic_socket_streambuf

Construct a `basic_socket_streambuf` without establishing a connection.

```
basic_socket_streambuf();
```

basic_socket_streambuf::bind

Bind the socket to the given local endpoint.

```
void bind(
    const endpoint_type & endpoint);
  » more...

boost::system::error_code bind(
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
  » more...
```

basic_socket_streambuf::bind (1 of 2 overloads)

Inherited from basic_socket.

Bind the socket to the given local endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void bind(
    const endpoint_type & endpoint);
```

This function binds the socket to the specified endpoint on the local machine.

Parameters

endpoint An endpoint on the local machine to which the socket will be bound.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
socket.bind(boost::asio::ip::tcp::endpoint(
      boost::asio::ip::tcp::v4(), 12345));
```

basic_socket_streambuf::bind (2 of 2 overloads)

Inherited from basic_socket.

Bind the socket to the given local endpoint.

```
boost::system::error_code bind(
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
```

This function binds the socket to the specified endpoint on the local machine.

Parameters

endpoint An endpoint on the local machine to which the socket will be bound.

ec Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
boost::system::error_code ec;
socket.bind(boost::asio::ip::tcp::endpoint(
      boost::asio::ip::tcp::v4(), 12345), ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_streambuf::broadcast

Inherited from socket_base.

Socket option to permit sending of broadcast messages.

```
typedef implementation_defined broadcast;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Implements the SOL_SOCKET/SO_BROADCAST socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::bytes_readable

Inherited from socket_base.

IO control command to get the amount of data that can be read without blocking.

```
typedef implementation_defined bytes_readable;
```

Implements the FIONREAD IO control command.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::bytes_readable command(true);
socket.io_control(command);
std::size_t bytes_readable = command.get();
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::cancel

Cancel all asynchronous operations associated with the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

basic_socket_streambuf::cancel (1 of 2 overloads)

Inherited from basic_socket.

Cancel all asynchronous operations associated with the socket.

```
void cancel();
```

This function causes all outstanding asynchronous connect, send and receive operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

Calls to `cancel()` will always fail with `boost::asio::error::operation_not_supported` when run on Windows XP, Windows Server 2003, and earlier versions of Windows, unless BOOST_ASIO_ENABLE_CANCELIO is defined. However, the CancelIo function has two issues that should be considered before enabling its use:

- It will only cancel asynchronous operations that were initiated in the current thread.
- It can appear to complete without error, but the request to cancel the unfinished operations may be silently ignored by the operating system. Whether it works or not seems to depend on the drivers that are installed.

For portable cancellation, consider using one of the following alternatives:

- Disable asio's I/O completion port backend by defining BOOST_ASIO_DISABLE_IOCP.
- Use the `close()` function to simultaneously cancel the outstanding operations and close the socket.

When running on Windows Vista, Windows Server 2008, and later, the CancelIoEx function is always used. This function does not have the problems described above.

basic_socket_streambuf::cancel (2 of 2 overloads)

Inherited from basic_socket.

Cancel all asynchronous operations associated with the socket.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function causes all outstanding asynchronous connect, send and receive operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Parameters

ec	Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

Calls to `cancel()` will always fail with `boost::asio::error::operation_not_supported` when run on Windows XP, Windows Server 2003, and earlier versions of Windows, unless BOOST_ASIO_ENABLE_CANCELIO is defined. However, the CancelIo function has two issues that should be considered before enabling its use:

- It will only cancel asynchronous operations that were initiated in the current thread.
- It can appear to complete without error, but the request to cancel the unfinished operations may be silently ignored by the operating system. Whether it works or not seems to depend on the drivers that are installed.

For portable cancellation, consider using one of the following alternatives:

- Disable asio's I/O completion port backend by defining BOOST_ASIO_DISABLE_IOCP.
- Use the `close()` function to simultaneously cancel the outstanding operations and close the socket.

When running on Windows Vista, Windows Server 2008, and later, the CancelIoEx function is always used. This function does not have the problems described above.

basic_socket_streambuf::close

Close the connection.

```
basic_socket_streambuf< Protocol, StreamSocketService, Time, TimeTraits, TimerService > * close();
  » more...
```

Close the socket.

```
boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

basic_socket_streambuf::close (1 of 2 overloads)

Close the connection.

```
basic_socket_streambuf< Protocol, StreamSocketService, Time, TimeTraits, TimerService > * close();
```

Return Value

`this` if a connection was successfully established, a null pointer otherwise.

basic_socket_streambuf::close (2 of 2 overloads)

Inherited from basic_socket.

Close the socket.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```

This function is used to close the socket. Any asynchronous send, receive or connect operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

ec Set to indicate what error occurred, if any. Note that, even if the function indicates an error, the underlying descriptor is closed.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
socket.close(ec);
if (ec)
{
  // An error occurred.
}
```

Remarks

For portable behaviour with respect to graceful closure of a connected socket, call `shutdown()` before closing the socket.

basic_socket_streambuf::connect

Establish a connection.

```
basic_socket_streambuf< Protocol, StreamSocketService, Time, TimeTraits, TimerService > * connect(
    const endpoint_type & endpoint);
  » more...

template<
    typename T1,
    ... ,
    typename TN>
basic_socket_streambuf< Protocol, StreamSocketService > * connect(
    T1 t1,
    ... ,
    TN tn);
  » more...
```

Connect the socket to the specified endpoint.

```
boost::system::error_code connect(
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
  » more...
```

basic_socket_streambuf::connect (1 of 3 overloads)

Establish a connection.

```
basic_socket_streambuf< Protocol, StreamSocketService, Time, TimeTraits, TimerService > * connect(
    const endpoint_type & endpoint);
```

This function establishes a connection to the specified endpoint.

Return Value

`this` if a connection was successfully established, a null pointer otherwise.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_streambuf::connect (2 of 3 overloads)

Establish a connection.

```
template<
    typename T1,
    ... ,
    typename TN>
basic_socket_streambuf< Protocol, StreamSocketService > * connect(
    T1 t1,
    ... ,
    TN tn);
```

This function automatically establishes a connection based on the supplied resolver query parameters. The arguments are used to construct a resolver query object.

Return Value

`this` if a connection was successfully established, a null pointer otherwise.

basic_socket_streambuf::connect (3 of 3 overloads)

Inherited from basic_socket.

Connect the socket to the specified endpoint.

```
boost::system::error_code connect(
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
```

This function is used to connect a socket to the specified remote endpoint. The function call will block until the connection is successfully made or an error occurs.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint	The remote endpoint to which the socket will be connected.
ec	Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
boost::system::error_code ec;
socket.connect(endpoint, ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_streambuf::debug

Inherited from socket_base.

Socket option to enable socket-level debugging.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef implementation_defined debug;
```

Implements the SOL_SOCKET/SO_DEBUG socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::do_not_route

Inherited from socket_base.

Socket option to prevent routing, use local interfaces only.

```
typedef implementation_defined do_not_route;
```

Implements the SOL_SOCKET/SO_DONTROUTE socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::duration_type

The duration type.

```
typedef TimeTraits::duration_type duration_type;
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::enable_connection_aborted

Inherited from socket_base.

Socket option to report aborted connections on accept.

```
typedef implementation_defined enable_connection_aborted;
```

Implements a custom socket option that determines whether or not an accept operation is permitted to fail with `boost::asio::error::connection_aborted`. By default the option is false.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option(true);
acceptor.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::endpoint_type

The endpoint type.

```
typedef Protocol::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::error

Get the last error associated with the stream buffer.

```
virtual const boost::system::error_code & error() const;
```

Return Value

An `error_code` corresponding to the last error from the stream buffer.

basic_socket_streambuf::expires_at

Get the stream buffer's expiry time as an absolute time.

```
time_type expires_at() const;
  » more...
```

Set the stream buffer's expiry time as an absolute time.

```
void expires_at(
    const time_type & expiry_time);
  » more...
```

basic_socket_streambuf::expires_at (1 of 2 overloads)

Get the stream buffer's expiry time as an absolute time.

```
time_type expires_at() const;
```

Return Value

An absolute time value representing the stream buffer's expiry time.

basic_socket_streambuf::expires_at (2 of 2 overloads)

Set the stream buffer's expiry time as an absolute time.

```
void expires_at(
    const time_type & expiry_time);
```

This function sets the expiry time associated with the stream. Stream operations performed after this time (where the operations cannot be completed using the internal buffers) will fail with the error `boost::asio::error::operation_aborted`.

Parameters

expiry_time	The expiry time to be used for the stream.

basic_socket_streambuf::expires_from_now

Get the stream buffer's expiry time relative to now.

```
duration_type expires_from_now() const;
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Set the stream buffer's expiry time relative to now.

```
void expires_from_now(
    const duration_type & expiry_time);
  » more...
```

basic_socket_streambuf::expires_from_now (1 of 2 overloads)

Get the stream buffer's expiry time relative to now.

```
duration_type expires_from_now() const;
```

Return Value

A relative time value representing the stream buffer's expiry time.

basic_socket_streambuf::expires_from_now (2 of 2 overloads)

Set the stream buffer's expiry time relative to now.

```
void expires_from_now(
    const duration_type & expiry_time);
```

This function sets the expiry time associated with the stream. Stream operations performed after this time (where the operations cannot be completed using the internal buffers) will fail with the error `boost::asio::error::operation_aborted`.

Parameters

expiry_time The expiry time to be used for the timer.

basic_socket_streambuf::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

basic_socket_streambuf::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

basic_socket_streambuf::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_streambuf::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

basic_socket_streambuf::get_option

Get an option from the socket.

```
void get_option(
    GettableSocketOption & option) const;
  » more...

boost::system::error_code get_option(
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
  » more...
```

basic_socket_streambuf::get_option (1 of 2 overloads)

Inherited from basic_socket.

Get an option from the socket.

```
template<
    typename GettableSocketOption>
void get_option(
    GettableSocketOption & option) const;
```

This function is used to get the current value of an option on the socket.

Parameters

option The option value to be obtained from the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Getting the value of the SOL_SOCKET/SO_KEEPALIVE option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::keep_alive option;
socket.get_option(option);
bool is_set = option.get();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_streambuf::get_option (2 of 2 overloads)

Inherited from basic_socket.

Get an option from the socket.

```
template<
    typename GettableSocketOption>
boost::system::error_code get_option(
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
```

This function is used to get the current value of an option on the socket.

Parameters

option The option value to be obtained from the socket.

ec Set to indicate what error occurred, if any.

Example

Getting the value of the SOL_SOCKET/SO_KEEPALIVE option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::keep_alive option;
boost::system::error_code ec;
socket.get_option(option, ec);
if (ec)
{
  // An error occurred.
}
bool is_set = option.get();
```

basic_socket_streambuf::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

basic_socket_streambuf::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

basic_socket_streambuf::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
const service_type & get_service() const;
```

basic_socket_streambuf::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

basic_socket_streambuf::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::io_control

Perform an IO control command on the socket.

```
void io_control(
    IoControlCommand & command);
  » more...

boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
  » more...
```

basic_socket_streambuf::io_control (1 of 2 overloads)

Inherited from basic_socket.

Perform an IO control command on the socket.

```
template<
    typename IoControlCommand>
void io_control(
    IoControlCommand & command);
```

This function is used to execute an IO control command on the socket.

Parameters

command The IO control command to be performed on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Example

Getting the number of bytes ready to read:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::bytes_readable command;
socket.io_control(command);
std::size_t bytes_readable = command.get();
```

basic_socket_streambuf::io_control (2 of 2 overloads)

Inherited from basic_socket.

Perform an IO control command on the socket.

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
```

This function is used to execute an IO control command on the socket.

Parameters

command	The IO control command to be performed on the socket.
ec	Set to indicate what error occurred, if any.

Example

Getting the number of bytes ready to read:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::bytes_readable command;
boost::system::error_code ec;
socket.io_control(command, ec);
if (ec)
{
  // An error occurred.
}
std::size_t bytes_readable = command.get();
```

basic_socket_streambuf::io_handler

```
friend struct io_handler();
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_streambuf::is_open

Inherited from basic_socket.

Determine whether the socket is open.

```
bool is_open() const;
```

basic_socket_streambuf::keep_alive

Inherited from socket_base.

Socket option to send keep-alives.

```
typedef implementation_defined keep_alive;
```

Implements the SOL_SOCKET/SO_KEEPALIVE socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::linger

Inherited from socket_base.

Socket option to specify whether the socket lingers on close if unsent data is present.

```
typedef implementation_defined linger;
```

Implements the SOL_SOCKET/SO_LINGER socket option.

Examples

Setting the option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option(true, 30);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option;
socket.get_option(option);
bool is_set = option.enabled();
unsigned short timeout = option.timeout();
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::local_endpoint

Get the local endpoint of the socket.

```
endpoint_type local_endpoint() const;
  » more...

endpoint_type local_endpoint(
    boost::system::error_code & ec) const;
  » more...
```

basic_socket_streambuf::local_endpoint (1 of 2 overloads)

Inherited from basic_socket.

Get the local endpoint of the socket.

```
endpoint_type local_endpoint() const;
```

This function is used to obtain the locally bound endpoint of the socket.

Return Value

An object that represents the local endpoint of the socket.

Exceptions

boost::system::system_error	Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::endpoint endpoint = socket.local_endpoint();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_streambuf::local_endpoint (2 of 2 overloads)

Inherited from basic_socket.

Get the local endpoint of the socket.

```
endpoint_type local_endpoint(
    boost::system::error_code & ec) const;
```

This function is used to obtain the locally bound endpoint of the socket.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

An object that represents the local endpoint of the socket. Returns a default-constructed endpoint object if an error occurred.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
boost::asio::ip::tcp::endpoint endpoint = socket.local_endpoint(ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_streambuf::lowest_layer

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

basic_socket_streambuf::lowest_layer (1 of 2 overloads)

Inherited from basic_socket.

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

This function returns a reference to the lowest layer in a stack of layers. Since a `basic_socket` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_streambuf::lowest_layer (2 of 2 overloads)

Inherited from basic_socket.

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

This function returns a const reference to the lowest layer in a stack of layers. Since a `basic_socket` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A const reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

basic_socket_streambuf::lowest_layer_type

Inherited from basic_socket.

A `basic_socket` is always the lowest layer.

```
typedef basic_socket< Protocol, StreamSocketService > lowest_layer_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_socket	Construct a basic_socket without opening it. Construct and open a basic_socket. Construct a basic_socket, opening it and binding it to the given local endpoint. Construct a basic_socket on an existing native socket. Move-construct a basic_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_socket from another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
remote_endpoint	Get the remote endpoint of the socket.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_socket	Protected destructor to prevent deletion through this type.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_socket` class template provides functionality that is common to both stream-oriented and datagram-oriented sockets.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_streambuf::max_connections

Inherited from socket_base.

The maximum length of the queue of pending incoming connections.

```
static const int max_connections = implementation_defined;
```

basic_socket_streambuf::message_do_not_route

Inherited from socket_base.

Specify that the data should not be subject to routing.

```
static const int message_do_not_route = implementation_defined;
```

basic_socket_streambuf::message_end_of_record

Inherited from socket_base.

Specifies that the data marks the end of a record.

```
static const int message_end_of_record = implementation_defined;
```

basic_socket_streambuf::message_flags

Inherited from socket_base.

Bitmask type for flags that can be passed to send and receive operations.

```
typedef int message_flags;
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::message_out_of_band

Inherited from socket_base.

Process out-of-band data.

```
static const int message_out_of_band = implementation_defined;
```

basic_socket_streambuf::message_peek

Inherited from socket_base.

Peek at incoming data without removing it from the input queue.

```
static const int message_peek = implementation_defined;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_streambuf::native

Inherited from basic_socket.

(Deprecated: Use `native_handle()`.) Get the native socket representation.

```
native_type native();
```

This function may be used to obtain the underlying representation of the socket. This is intended to allow access to native socket functionality that is not otherwise provided.

basic_socket_streambuf::native_handle

Inherited from basic_socket.

Get the native socket representation.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying representation of the socket. This is intended to allow access to native socket functionality that is not otherwise provided.

basic_socket_streambuf::native_handle_type

Inherited from basic_socket.

The native representation of a socket.

```
typedef StreamSocketService::native_handle_type native_handle_type;
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::native_non_blocking

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking() const;
  » more...
```

Sets the non-blocking mode of the native socket implementation.

```
void native_non_blocking(
    bool mode);
  » more...

boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_streambuf::native_non_blocking (1 of 3 overloads)

Inherited from basic_socket.

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking() const;
```

This function is used to retrieve the non-blocking mode of the underlying native socket. This mode has no effect on the behaviour of the socket object's synchronous operations.

Return Value

`true` if the underlying socket is in non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Remarks

The current non-blocking mode is cached by the socket object. Consequently, the return value may be incorrect if the non-blocking mode was set directly on the native socket.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_socket_streambuf::native_non_blocking (2 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the native socket implementation.

```
void native_non_blocking(
    bool mode);
```

This function is used to modify the non-blocking mode of the underlying native socket. It has no effect on the behaviour of the socket object's synchronous operations.

Parameters

mode — If `true`, the underlying socket is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Exceptions

boost::system::system_error — Thrown on failure. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_socket_streambuf::native_non_blocking (3 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the native socket implementation.

```
boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

This function is used to modify the non-blocking mode of the underlying native socket. It has no effect on the behaviour of the socket object's synchronous operations.

Parameters

mode — If `true`, the underlying socket is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

ec — Set to indicate what error occurred, if any. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_socket_streambuf::native_type

Inherited from basic_socket.

(Deprecated: Use native_handle_type.) The native representation of a socket.

```
typedef StreamSocketService::native_handle_type native_type;
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::non_blocking

Gets the non-blocking mode of the socket.

```
bool non_blocking() const;
  » more...
```

Sets the non-blocking mode of the socket.

```
void non_blocking(
    bool mode);
  » more...

boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

basic_socket_streambuf::non_blocking (1 of 3 overloads)

Inherited from basic_socket.

Gets the non-blocking mode of the socket.

```
bool non_blocking() const;
```

Return Value

`true` if the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_streambuf::non_blocking (2 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the socket.

```
void non_blocking(
    bool mode);
```

Parameters

mode — If `true`, the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Exceptions

boost::system::system_error — Thrown on failure.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_socket_streambuf::non_blocking (3 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the socket.

```
boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

Parameters

mode — If `true`, the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

ec — Set to indicate what error occurred, if any.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_socket_streambuf::non_blocking_io

Inherited from socket_base.

(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.

```
typedef implementation_defined non_blocking_io;
```

Implements the FIONBIO IO control command.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::non_blocking_io command(true);
socket.io_control(command);
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::open

Open the socket using the specified protocol.

```
void open(
    const protocol_type & protocol = protocol_type());
  » more...

boost::system::error_code open(
    const protocol_type & protocol,
    boost::system::error_code & ec);
  » more...
```

basic_socket_streambuf::open (1 of 2 overloads)

Inherited from basic_socket.

Open the socket using the specified protocol.

```
void open(
    const protocol_type & protocol = protocol_type());
```

This function opens the socket so that it will use the specified protocol.

Parameters

protocol An object specifying protocol parameters to be used.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
```

basic_socket_streambuf::open (2 of 2 overloads)

Inherited from basic_socket.

Open the socket using the specified protocol.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code open(
    const protocol_type & protocol,
    boost::system::error_code & ec);
```

This function opens the socket so that it will use the specified protocol.

Parameters

protocol	An object specifying which protocol is to be used.
ec	Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::system::error_code ec;
socket.open(boost::asio::ip::tcp::v4(), ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_streambuf::overflow

```
int_type overflow(
    int_type c);
```

basic_socket_streambuf::protocol_type

Inherited from basic_socket.

The protocol type.

```
typedef Protocol protocol_type;
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::puberror

Get the last error associated with the stream buffer.

```
const boost::system::error_code & puberror() const;
```

Return Value

An `error_code` corresponding to the last error from the stream buffer.

basic_socket_streambuf::receive_buffer_size

Inherited from socket_base.

Socket option for the receive buffer size of a socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef implementation_defined receive_buffer_size;
```

Implements the SOL_SOCKET/SO_RCVBUF socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::receive_low_watermark

Inherited from socket_base.

Socket option for the receive low watermark.

```
typedef implementation_defined receive_low_watermark;
```

Implements the SOL_SOCKET/SO_RCVLOWAT socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::remote_endpoint

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint() const;
  » more...

endpoint_type remote_endpoint(
    boost::system::error_code & ec) const;
  » more...
```

basic_socket_streambuf::remote_endpoint (1 of 2 overloads)

Inherited from basic_socket.

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint() const;
```

This function is used to obtain the remote endpoint of the socket.

Return Value

An object that represents the remote endpoint of the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::endpoint endpoint = socket.remote_endpoint();
```

basic_socket_streambuf::remote_endpoint (2 of 2 overloads)

Inherited from basic_socket.

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint(
    boost::system::error_code & ec) const;
```

This function is used to obtain the remote endpoint of the socket.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

An object that represents the remote endpoint of the socket. Returns a default-constructed endpoint object if an error occurred.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
boost::asio::ip::tcp::endpoint endpoint = socket.remote_endpoint(ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_streambuf::reuse_address

Inherited from socket_base.

Socket option to allow the socket to be bound to an address that is already in use.

```
typedef implementation_defined reuse_address;
```

Implements the SOL_SOCKET/SO_REUSEADDR socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option(true);
acceptor.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::send_buffer_size

Inherited from socket_base.

Socket option for the send buffer size of a socket.

```
typedef implementation_defined send_buffer_size;
```

Implements the SOL_SOCKET/SO_SNDBUF socket option.

Examples

Setting the option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::send_low_watermark

Inherited from socket_base.

Socket option for the send low watermark.

```
typedef implementation_defined send_low_watermark;
```

Implements the SOL_SOCKET/SO_SNDLOWAT socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

basic_socket_streambuf::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef StreamSocketService service_type;
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::set_option

Set an option on the socket.

```
void set_option(
    const SettableSocketOption & option);
  » more...

boost::system::error_code set_option(
    const SettableSocketOption & option,
    boost::system::error_code & ec);
  » more...
```

basic_socket_streambuf::set_option (1 of 2 overloads)

Inherited from basic_socket.

Set an option on the socket.

```
template<
    typename SettableSocketOption>
void set_option(
    const SettableSocketOption & option);
```

This function is used to set an option on the socket.

Parameters

option The new option value to be set on the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Setting the IPPROTO_TCP/TCP_NODELAY option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::no_delay option(true);
socket.set_option(option);
```

basic_socket_streambuf::set_option (2 of 2 overloads)

Inherited from basic_socket.

Set an option on the socket.

```
template<
    typename SettableSocketOption>
boost::system::error_code set_option(
    const SettableSocketOption & option,
    boost::system::error_code & ec);
```

This function is used to set an option on the socket.

Parameters

option The new option value to be set on the socket.

ec Set to indicate what error occurred, if any.

Example

Setting the IPPROTO_TCP/TCP_NODELAY option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::no_delay option(true);
boost::system::error_code ec;
socket.set_option(option, ec);
if (ec)
{
  // An error occurred.
}
```

basic_socket_streambuf::setbuf

```
std::streambuf * setbuf(
    char_type * s,
    std::streamsize n);
```

basic_socket_streambuf::shutdown

Disable sends or receives on the socket.

```
void shutdown(
    shutdown_type what);
  » more...

boost::system::error_code shutdown(
    shutdown_type what,
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_streambuf::shutdown (1 of 2 overloads)

Inherited from basic_socket.

Disable sends or receives on the socket.

```
void shutdown(
    shutdown_type what);
```

This function is used to disable send operations, receive operations, or both.

Parameters

what Determines what types of operation will no longer be allowed.

Exceptions

boost::system::system_error Thrown on failure.

Example

Shutting down the send side of the socket:

```
boost::asio::ip::tcp::socket socket(io_service);
...
socket.shutdown(boost::asio::ip::tcp::socket::shutdown_send);
```

basic_socket_streambuf::shutdown (2 of 2 overloads)

Inherited from basic_socket.

Disable sends or receives on the socket.

```
boost::system::error_code shutdown(
    shutdown_type what,
    boost::system::error_code & ec);
```

This function is used to disable send operations, receive operations, or both.

Parameters

what Determines what types of operation will no longer be allowed.

ec Set to indicate what error occurred, if any.

Example

Shutting down the send side of the socket:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
socket.shutdown(boost::asio::ip::tcp::socket::shutdown_send, ec);
if (ec)
{
  // An error occurred.
}
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_streambuf::shutdown_type

Inherited from socket_base.

Different ways a socket may be shutdown.

```
enum shutdown_type
```

Values

shutdown_receive	Shutdown the receive side of the socket.
shutdown_send	Shutdown the send side of the socket.
shutdown_both	Shutdown both send and receive on the socket.

basic_socket_streambuf::sync

```
int sync();
```

basic_socket_streambuf::time_type

The time type.

```
typedef TimeTraits::time_type time_type;
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::timer_handler

```
friend struct timer_handler();
```

Requirements

Header: `boost/asio/basic_socket_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_socket_streambuf::underflow

```
int_type underflow();
```

basic_socket_streambuf::~basic_socket_streambuf

Destructor flushes buffered data.

```
virtual ~basic_socket_streambuf();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_stream_socket

Provides stream-oriented socket functionality.

```
template<
    typename Protocol,
    typename StreamSocketService = stream_socket_service<Protocol>>
class basic_stream_socket :
  public basic_socket< Protocol, StreamSocketService >
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
async_read_some	Start an asynchronous read.
async_receive	Start an asynchronous receive.
async_send	Start an asynchronous send.
async_write_some	Start an asynchronous write.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_stream_socket	Construct a basic_stream_socket without opening it. Construct and open a basic_stream_socket. Construct a basic_stream_socket, opening it and binding it to the given local endpoint. Construct a basic_stream_socket on an existing native socket. Move-construct a basic_stream_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_stream_socket from another.
read_some	Read some data from the socket.
receive	Receive some data on the socket. Receive some data on a connected socket.
remote_endpoint	Get the remote endpoint of the socket.
send	Send some data on the socket.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.
write_some	Write some data to the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_stream_socket` class template provides asynchronous and blocking stream-oriented socket functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::assign

Assign an existing native socket to the socket.

```
void assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket);
  » more...

boost::system::error_code assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::assign (1 of 2 overloads)

Inherited from basic_socket.

Assign an existing native socket to the socket.

```
void assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket);
```

basic_stream_socket::assign (2 of 2 overloads)

Inherited from basic_socket.

Assign an existing native socket to the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code assign(
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
```

basic_stream_socket::async_connect

Inherited from basic_socket.

Start an asynchronous connect.

```
template<
    typename ConnectHandler>
void async_connect(
    const endpoint_type & peer_endpoint,
    ConnectHandler handler);
```

This function is used to asynchronously connect a socket to the specified remote endpoint. The function call always returns immediately.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint — The remote endpoint to which the socket will be connected. Copies will be made of the endpoint object as required.

handler — The handler to be called when the connection operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error // Result of operation
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Example

```
void connect_handler(const boost::system::error_code& error)
{
  if (!error)
  {
    // Connect succeeded.
  }
}

...

boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.async_connect(endpoint, connect_handler);
```

basic_stream_socket::async_read_some

Start an asynchronous read.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_some(
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

This function is used to asynchronously read data from the stream socket. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be read. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes read.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The read operation may not read all of the requested number of bytes. Consider using the `async_read` function if you need to ensure that the requested amount of data is read before the asynchronous operation completes.

Example

To read into a single data buffer use the `buffer` function as follows:

```
socket.async_read_some(boost::asio::buffer(data, size), handler);
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_stream_socket::async_receive

Start an asynchronous receive.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    ReadHandler handler);
  » more...

template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    ReadHandler handler);
  » more...
```

basic_stream_socket::async_receive (1 of 2 overloads)

Start an asynchronous receive.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

This function is used to asynchronously receive data from the stream socket. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be received. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the receive operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes received.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The receive operation may not receive all of the requested number of bytes. Consider using the `async_read` function if you need to ensure that the requested amount of data is received before the asynchronous operation completes.

Example

To receive into a single data buffer use the `buffer` function as follows:

```
socket.async_receive(boost::asio::buffer(data, size), handler);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

See the buffer documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_stream_socket::async_receive (2 of 2 overloads)

Start an asynchronous receive.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    ReadHandler handler);
```

This function is used to asynchronously receive data from the stream socket. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be received. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

flags — Flags specifying how the receive call is to be made.

handler — The handler to be called when the receive operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes received.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The receive operation may not receive all of the requested number of bytes. Consider using the async_read function if you need to ensure that the requested amount of data is received before the asynchronous operation completes.

Example

To receive into a single data buffer use the buffer function as follows:

```
socket.async_receive(boost::asio::buffer(data, size), 0, handler);
```

See the buffer documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_stream_socket::async_send

Start an asynchronous send.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    const ConstBufferSequence & buffers,
    WriteHandler handler);
  » more...

template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    WriteHandler handler);
  » more...
```

basic_stream_socket::async_send (1 of 2 overloads)

Start an asynchronous send.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

This function is used to asynchronously send data on the stream socket. The function call always returns immediately.

Parameters

buffers — One or more data buffers to be sent on the socket. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the send operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes sent.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The send operation may not transmit all of the data to the peer. Consider using the `async_write` function if you need to ensure that all data is written before the asynchronous operation completes.

Example

To send a single data buffer use the `buffer` function as follows:

```
socket.async_send(boost::asio::buffer(data, size), handler);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

See the buffer documentation for information on sending multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_stream_socket::async_send (2 of 2 overloads)

Start an asynchronous send.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    WriteHandler handler);
```

This function is used to asynchronously send data on the stream socket. The function call always returns immediately.

Parameters

buffers — One or more data buffers to be sent on the socket. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

flags — Flags specifying how the send call is to be made.

handler — The handler to be called when the send operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes sent.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using boost::asio::io_service::post().

Remarks

The send operation may not transmit all of the data to the peer. Consider using the async_write function if you need to ensure that all data is written before the asynchronous operation completes.

Example

To send a single data buffer use the buffer function as follows:

```
socket.async_send(boost::asio::buffer(data, size), 0, handler);
```

See the buffer documentation for information on sending multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_stream_socket::async_write_some

Start an asynchronous write.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_some(
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

This function is used to asynchronously write data to the stream socket. The function call always returns immediately.

Parameters

buffers — One or more data buffers to be written to the socket. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the write operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes written.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The write operation may not transmit all of the data to the peer. Consider using the `async_write` function if you need to ensure that all data is written before the asynchronous operation completes.

Example

To write a single data buffer use the `buffer` function as follows:

```
socket.async_write_some(boost::asio::buffer(data, size), handler);
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_stream_socket::at_mark

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark() const;
  » more...

bool at_mark(
    boost::system::error_code & ec) const;
  » more...
```

basic_stream_socket::at_mark (1 of 2 overloads)

Inherited from basic_socket.

Determine whether the socket is at the out-of-band data mark.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
bool at_mark() const;
```

This function is used to check whether the socket input is currently positioned at the out-of-band data mark.

Return Value

A bool indicating whether the socket is at the out-of-band data mark.

Exceptions

boost::system::system_error Thrown on failure.

basic_stream_socket::at_mark (2 of 2 overloads)

Inherited from basic_socket.

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark(
    boost::system::error_code & ec) const;
```

This function is used to check whether the socket input is currently positioned at the out-of-band data mark.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

A bool indicating whether the socket is at the out-of-band data mark.

basic_stream_socket::available

Determine the number of bytes available for reading.

```
std::size_t available() const;
  » more...

std::size_t available(
    boost::system::error_code & ec) const;
  » more...
```

basic_stream_socket::available (1 of 2 overloads)

Inherited from basic_socket.

Determine the number of bytes available for reading.

```
std::size_t available() const;
```

This function is used to determine the number of bytes that may be read without blocking.

Return Value

The number of bytes that may be read without blocking, or 0 if an error occurs.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

basic_stream_socket::available (2 of 2 overloads)

Inherited from basic_socket.

Determine the number of bytes available for reading.

```
std::size_t available(
    boost::system::error_code & ec) const;
```

This function is used to determine the number of bytes that may be read without blocking.

Parameters

ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes that may be read without blocking, or 0 if an error occurs.

basic_stream_socket::basic_stream_socket

Construct a `basic_stream_socket` without opening it.

```
explicit basic_stream_socket(
    boost::asio::io_service & io_service);
  » more...
```

Construct and open a `basic_stream_socket`.

```
basic_stream_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol);
  » more...
```

Construct a `basic_stream_socket`, opening it and binding it to the given local endpoint.

```
basic_stream_socket(
    boost::asio::io_service & io_service,
    const endpoint_type & endpoint);
  » more...
```

Construct a `basic_stream_socket` on an existing native socket.

```
basic_stream_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol,
    const native_handle_type & native_socket);
  » more...
```

Move-construct a `basic_stream_socket` from another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
basic_stream_socket(
    basic_stream_socket && other);
  » more...
```

basic_stream_socket::basic_stream_socket (1 of 5 overloads)

Construct a `basic_stream_socket` without opening it.

```
basic_stream_socket(
    boost::asio::io_service & io_service);
```

This constructor creates a stream socket without opening it. The socket needs to be opened and then connected or accepted before data can be sent or received on it.

Parameters

io_service — The `io_service` object that the stream socket will use to dispatch handlers for any asynchronous operations performed on the socket.

basic_stream_socket::basic_stream_socket (2 of 5 overloads)

Construct and open a `basic_stream_socket`.

```
basic_stream_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol);
```

This constructor creates and opens a stream socket. The socket needs to be connected or accepted before data can be sent or received on it.

Parameters

io_service — The `io_service` object that the stream socket will use to dispatch handlers for any asynchronous operations performed on the socket.

protocol — An object specifying protocol parameters to be used.

Exceptions

boost::system::system_error — Thrown on failure.

basic_stream_socket::basic_stream_socket (3 of 5 overloads)

Construct a `basic_stream_socket`, opening it and binding it to the given local endpoint.

```
basic_stream_socket(
    boost::asio::io_service & io_service,
    const endpoint_type & endpoint);
```

This constructor creates a stream socket and automatically opens it bound to the specified endpoint on the local machine. The protocol used is the protocol associated with the given endpoint.

Parameters

io_service — The `io_service` object that the stream socket will use to dispatch handlers for any asynchronous operations performed on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

endpoint An endpoint on the local machine to which the stream socket will be bound.

Exceptions

boost::system::system_error Thrown on failure.

basic_stream_socket::basic_stream_socket (4 of 5 overloads)

Construct a `basic_stream_socket` on an existing native socket.

```
basic_stream_socket(
    boost::asio::io_service & io_service,
    const protocol_type & protocol,
    const native_handle_type & native_socket);
```

This constructor creates a stream socket object to hold an existing native socket.

Parameters

io_service The `io_service` object that the stream socket will use to dispatch handlers for any asynchronous operations performed on the socket.

protocol An object specifying protocol parameters to be used.

native_socket The new underlying socket implementation.

Exceptions

boost::system::system_error Thrown on failure.

basic_stream_socket::basic_stream_socket (5 of 5 overloads)

Move-construct a `basic_stream_socket` from another.

```
basic_stream_socket(
    basic_stream_socket && other);
```

This constructor moves a stream socket from one object to another.

Parameters

other The other `basic_stream_socket` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_stream_socket(io_service&)` constructor.

basic_stream_socket::bind

Bind the socket to the given local endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void bind(
    const endpoint_type & endpoint);
  » more...

boost::system::error_code bind(
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::bind (1 of 2 overloads)

Inherited from basic_socket.

Bind the socket to the given local endpoint.

```
void bind(
    const endpoint_type & endpoint);
```

This function binds the socket to the specified endpoint on the local machine.

Parameters

endpoint An endpoint on the local machine to which the socket will be bound.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
socket.bind(boost::asio::ip::tcp::endpoint(
      boost::asio::ip::tcp::v4(), 12345));
```

basic_stream_socket::bind (2 of 2 overloads)

Inherited from basic_socket.

Bind the socket to the given local endpoint.

```
boost::system::error_code bind(
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
```

This function binds the socket to the specified endpoint on the local machine.

Parameters

endpoint An endpoint on the local machine to which the socket will be bound.

ec Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
boost::system::error_code ec;
socket.bind(boost::asio::ip::tcp::endpoint(
      boost::asio::ip::tcp::v4(), 12345), ec);
if (ec)
{
  // An error occurred.
}
```

basic_stream_socket::broadcast

Inherited from socket_base.

Socket option to permit sending of broadcast messages.

```
typedef implementation_defined broadcast;
```

Implements the SOL_SOCKET/SO_BROADCAST socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::bytes_readable

Inherited from socket_base.

IO control command to get the amount of data that can be read without blocking.

```
typedef implementation_defined bytes_readable;
```

Implements the FIONREAD IO control command.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::bytes_readable command(true);
socket.io_control(command);
std::size_t bytes_readable = command.get();
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::cancel

Cancel all asynchronous operations associated with the socket.

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::cancel (1 of 2 overloads)

Inherited from basic_socket.

Cancel all asynchronous operations associated with the socket.

```
void cancel();
```

This function causes all outstanding asynchronous connect, send and receive operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

Calls to `cancel()` will always fail with `boost::asio::error::operation_not_supported` when run on Windows XP, Windows Server 2003, and earlier versions of Windows, unless BOOST_ASIO_ENABLE_CANCELIO is defined. However, the CancelIo function has two issues that should be considered before enabling its use:

- It will only cancel asynchronous operations that were initiated in the current thread.
- It can appear to complete without error, but the request to cancel the unfinished operations may be silently ignored by the operating system. Whether it works or not seems to depend on the drivers that are installed.

For portable cancellation, consider using one of the following alternatives:

- Disable asio's I/O completion port backend by defining BOOST_ASIO_DISABLE_IOCP.
- Use the `close()` function to simultaneously cancel the outstanding operations and close the socket.

When running on Windows Vista, Windows Server 2008, and later, the CancelIoEx function is always used. This function does not have the problems described above.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_stream_socket::cancel (2 of 2 overloads)

Inherited from basic_socket.

Cancel all asynchronous operations associated with the socket.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function causes all outstanding asynchronous connect, send and receive operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Parameters

ec Set to indicate what error occurred, if any.

Remarks

Calls to `cancel()` will always fail with `boost::asio::error::operation_not_supported` when run on Windows XP, Windows Server 2003, and earlier versions of Windows, unless BOOST_ASIO_ENABLE_CANCELIO is defined. However, the CancelIo function has two issues that should be considered before enabling its use:

- It will only cancel asynchronous operations that were initiated in the current thread.
- It can appear to complete without error, but the request to cancel the unfinished operations may be silently ignored by the operating system. Whether it works or not seems to depend on the drivers that are installed.

For portable cancellation, consider using one of the following alternatives:

- Disable asio's I/O completion port backend by defining BOOST_ASIO_DISABLE_IOCP.
- Use the `close()` function to simultaneously cancel the outstanding operations and close the socket.

When running on Windows Vista, Windows Server 2008, and later, the CancelIoEx function is always used. This function does not have the problems described above.

basic_stream_socket::close

Close the socket.

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::close (1 of 2 overloads)

Inherited from basic_socket.

Close the socket.

```
void close();
```

This function is used to close the socket. Any asynchronous send, receive or connect operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure. Note that, even if the function indicates an error, the underlying descriptor is closed.

Remarks

For portable behaviour with respect to graceful closure of a connected socket, call `shutdown()` before closing the socket.

basic_stream_socket::close (2 of 2 overloads)

Inherited from basic_socket.

Close the socket.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```

This function is used to close the socket. Any asynchronous send, receive or connect operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Parameters

ec Set to indicate what error occurred, if any. Note that, even if the function indicates an error, the underlying descriptor is closed.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
socket.close(ec);
if (ec)
{
  // An error occurred.
}
```

Remarks

For portable behaviour with respect to graceful closure of a connected socket, call `shutdown()` before closing the socket.

basic_stream_socket::connect

Connect the socket to the specified endpoint.

```
void connect(
    const endpoint_type & peer_endpoint);
  » more...

boost::system::error_code connect(
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::connect (1 of 2 overloads)

Inherited from basic_socket.

Connect the socket to the specified endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void connect(
    const endpoint_type & peer_endpoint);
```

This function is used to connect a socket to the specified remote endpoint. The function call will block until the connection is successfully made or an error occurs.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint	The remote endpoint to which the socket will be connected.

Exceptions

boost::system::system_error	Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
socket.connect(endpoint);
```

basic_stream_socket::connect (2 of 2 overloads)

Inherited from basic_socket.

Connect the socket to the specified endpoint.

```
boost::system::error_code connect(
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
```

This function is used to connect a socket to the specified remote endpoint. The function call will block until the connection is successfully made or an error occurs.

The socket is automatically opened if it is not already open. If the connect fails, and the socket was automatically opened, the socket is not returned to the closed state.

Parameters

peer_endpoint	The remote endpoint to which the socket will be connected.
ec	Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::asio::ip::tcp::endpoint endpoint(
    boost::asio::ip::address::from_string("1.2.3.4"), 12345);
boost::system::error_code ec;
socket.connect(endpoint, ec);
if (ec)
{
  // An error occurred.
}
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_stream_socket::debug

Inherited from socket_base.

Socket option to enable socket-level debugging.

```
typedef implementation_defined debug;
```

Implements the SOL_SOCKET/SO_DEBUG socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::do_not_route

Inherited from socket_base.

Socket option to prevent routing, use local interfaces only.

```
typedef implementation_defined do_not_route;
```

Implements the SOL_SOCKET/SO_DONTROUTE socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option(true);
socket.set_option(option);
```

Getting the current option value:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::enable_connection_aborted

Inherited from socket_base.

Socket option to report aborted connections on accept.

```
typedef implementation_defined enable_connection_aborted;
```

Implements a custom socket option that determines whether or not an accept operation is permitted to fail with `boost::asio::error::connection_aborted`. By default the option is false.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option(true);
acceptor.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::endpoint_type

The endpoint type.

```
typedef Protocol::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_stream_socket::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

basic_stream_socket::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

basic_stream_socket::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

basic_stream_socket::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

basic_stream_socket::get_option

Get an option from the socket.

```
void get_option(
    GettableSocketOption & option) const;
  » more...

boost::system::error_code get_option(
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
  » more...
```

basic_stream_socket::get_option (1 of 2 overloads)

Inherited from basic_socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Get an option from the socket.

```
template<
    typename GettableSocketOption>
void get_option(
    GettableSocketOption & option) const;
```

This function is used to get the current value of an option on the socket.

Parameters

option The option value to be obtained from the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Getting the value of the SOL_SOCKET/SO_KEEPALIVE option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::keep_alive option;
socket.get_option(option);
bool is_set = option.get();
```

basic_stream_socket::get_option (2 of 2 overloads)

Inherited from basic_socket.

Get an option from the socket.

```
template<
    typename GettableSocketOption>
boost::system::error_code get_option(
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
```

This function is used to get the current value of an option on the socket.

Parameters

option The option value to be obtained from the socket.

ec Set to indicate what error occurred, if any.

Example

Getting the value of the SOL_SOCKET/SO_KEEPALIVE option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::keep_alive option;
boost::system::error_code ec;
socket.get_option(option, ec);
if (ec)
{
  // An error occurred.
}
bool is_set = option.get();
```

basic_stream_socket::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

basic_stream_socket::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

basic_stream_socket::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

basic_stream_socket::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

basic_stream_socket::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::io_control

Perform an IO control command on the socket.

```
void io_control(
    IoControlCommand & command);
  » more...

boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::io_control (1 of 2 overloads)

Inherited from basic_socket.

Perform an IO control command on the socket.

```
template<
    typename IoControlCommand>
void io_control(
    IoControlCommand & command);
```

This function is used to execute an IO control command on the socket.

Parameters

command The IO control command to be performed on the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Getting the number of bytes ready to read:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::bytes_readable command;
socket.io_control(command);
std::size_t bytes_readable = command.get();
```

basic_stream_socket::io_control (2 of 2 overloads)

Inherited from basic_socket.

Perform an IO control command on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
```

This function is used to execute an IO control command on the socket.

Parameters

command The IO control command to be performed on the socket.

ec Set to indicate what error occurred, if any.

Example

Getting the number of bytes ready to read:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::socket::bytes_readable command;
boost::system::error_code ec;
socket.io_control(command, ec);
if (ec)
{
  // An error occurred.
}
std::size_t bytes_readable = command.get();
```

basic_stream_socket::is_open

Inherited from basic_socket.

Determine whether the socket is open.

```
bool is_open() const;
```

basic_stream_socket::keep_alive

Inherited from socket_base.

Socket option to send keep-alives.

```
typedef implementation_defined keep_alive;
```

Implements the SOL_SOCKET/SO_KEEPALIVE socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option(true);
socket.set_option(option);
```

Getting the current option value:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::linger

Inherited from socket_base.

Socket option to specify whether the socket lingers on close if unsent data is present.

```
typedef implementation_defined linger;
```

Implements the SOL_SOCKET/SO_LINGER socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option(true, 30);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option;
socket.get_option(option);
bool is_set = option.enabled();
unsigned short timeout = option.timeout();
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::local_endpoint

Get the local endpoint of the socket.

```
endpoint_type local_endpoint() const;
  » more...

endpoint_type local_endpoint(
    boost::system::error_code & ec) const;
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_stream_socket::local_endpoint (1 of 2 overloads)

Inherited from basic_socket.

Get the local endpoint of the socket.

```
endpoint_type local_endpoint() const;
```

This function is used to obtain the locally bound endpoint of the socket.

Return Value

An object that represents the local endpoint of the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::endpoint endpoint = socket.local_endpoint();
```

basic_stream_socket::local_endpoint (2 of 2 overloads)

Inherited from basic_socket.

Get the local endpoint of the socket.

```
endpoint_type local_endpoint(
    boost::system::error_code & ec) const;
```

This function is used to obtain the locally bound endpoint of the socket.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

An object that represents the local endpoint of the socket. Returns a default-constructed endpoint object if an error occurred.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
boost::asio::ip::tcp::endpoint endpoint = socket.local_endpoint(ec);
if (ec)
{
  // An error occurred.
}
```

basic_stream_socket::lowest_layer

Get a reference to the lowest layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

basic_stream_socket::lowest_layer (1 of 2 overloads)

Inherited from basic_socket.

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

This function returns a reference to the lowest layer in a stack of layers. Since a `basic_socket` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

basic_stream_socket::lowest_layer (2 of 2 overloads)

Inherited from basic_socket.

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

This function returns a const reference to the lowest layer in a stack of layers. Since a `basic_socket` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A const reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

basic_stream_socket::lowest_layer_type

Inherited from basic_socket.

A `basic_socket` is always the lowest layer.

```
typedef basic_socket< Protocol, StreamSocketService > lowest_layer_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_socket	Construct a basic_socket without opening it. Construct and open a basic_socket. Construct a basic_socket, opening it and binding it to the given local endpoint. Construct a basic_socket on an existing native socket. Move-construct a basic_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_socket from another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
remote_endpoint	Get the remote endpoint of the socket.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_socket	Protected destructor to prevent deletion through this type.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_socket` class template provides functionality that is common to both stream-oriented and datagram-oriented sockets.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_stream_socket::max_connections

Inherited from socket_base.

The maximum length of the queue of pending incoming connections.

```
static const int max_connections = implementation_defined;
```

basic_stream_socket::message_do_not_route

Inherited from socket_base.

Specify that the data should not be subject to routing.

```
static const int message_do_not_route = implementation_defined;
```

basic_stream_socket::message_end_of_record

Inherited from socket_base.

Specifies that the data marks the end of a record.

```
static const int message_end_of_record = implementation_defined;
```

basic_stream_socket::message_flags

Inherited from socket_base.

Bitmask type for flags that can be passed to send and receive operations.

```
typedef int message_flags;
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::message_out_of_band

Inherited from socket_base.

Process out-of-band data.

```
static const int message_out_of_band = implementation_defined;
```

basic_stream_socket::message_peek

Inherited from socket_base.

Peek at incoming data without removing it from the input queue.

```
static const int message_peek = implementation_defined;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_stream_socket::native

Inherited from basic_socket.

(Deprecated: Use `native_handle()`.) Get the native socket representation.

```
native_type native();
```

This function may be used to obtain the underlying representation of the socket. This is intended to allow access to native socket functionality that is not otherwise provided.

basic_stream_socket::native_handle

Inherited from basic_socket.

Get the native socket representation.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying representation of the socket. This is intended to allow access to native socket functionality that is not otherwise provided.

basic_stream_socket::native_handle_type

The native representation of a socket.

```
typedef StreamSocketService::native_handle_type native_handle_type;
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::native_non_blocking

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking() const;
  » more...
```

Sets the non-blocking mode of the native socket implementation.

```
void native_non_blocking(
    bool mode);
  » more...

boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::native_non_blocking (1 of 3 overloads)

Inherited from basic_socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking() const;
```

This function is used to retrieve the non-blocking mode of the underlying native socket. This mode has no effect on the behaviour of the socket object's synchronous operations.

Return Value

`true` if the underlying socket is in non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Remarks

The current non-blocking mode is cached by the socket object. Consequently, the return value may be incorrect if the non-blocking mode was set directly on the native socket.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_stream_socket::native_non_blocking (2 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the native socket implementation.

```
void native_non_blocking(
    bool mode);
```

This function is used to modify the non-blocking mode of the underlying native socket. It has no effect on the behaviour of the socket object's synchronous operations.

Parameters

mode — If `true`, the underlying socket is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Exceptions

boost::system::system_error — Thrown on failure. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_stream_socket::native_non_blocking (3 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the native socket implementation.

```
boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

This function is used to modify the non-blocking mode of the underlying native socket. It has no effect on the behaviour of the socket object's synchronous operations.

Parameters

mode — If `true`, the underlying socket is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

ec — Set to indicate what error occurred, if any. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

Example

This function is intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The following example illustrates how Linux's `sendfile` system call might be encapsulated:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
struct sendfile_op
{
  tcp::socket& sock_;
  int fd_;
  Handler handler_;
  off_t offset_;
  std::size_t total_bytes_transferred_;

  // Function call operator meeting WriteHandler requirements.
  // Used as the handler for the async_write_some operation.
  void operator()(boost::system::error_code ec, std::size_t)
  {
    // Put the underlying socket into non-blocking mode.
    if (!ec)
      if (!sock_.native_non_blocking())
        sock_.native_non_blocking(true, ec);

    if (!ec)
    {
      for (;;)
      {
        // Try the system call.
        errno = 0;
        int n = ::sendfile(sock_.native_handle(), fd_, &offset_, 65536);
        ec = boost::system::error_code(n < 0 ? errno : 0,
            boost::asio::error::get_system_category());
        total_bytes_transferred_ += ec ? 0 : n;

        // Retry operation immediately if interrupted by signal.
        if (ec == boost::asio::error::interrupted)
          continue;

        // Check if we need to run the operation again.
        if (ec == boost::asio::error::would_block
            || ec == boost::asio::error::try_again)
        {
          // We have to wait for the socket to become ready again.
          sock_.async_write_some(boost::asio::null_buffers(), *this);
          return;
        }

        if (ec || n == 0)
        {
          // An error occurred, or we have reached the end of the file.
          // Either way we must exit the loop so we can call the handler.
          break;
        }

        // Loop around to try calling sendfile again.
      }
    }

    // Pass result back to user's handler.
    handler_(ec, total_bytes_transferred_);
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template <typename Handler>
void async_sendfile(tcp::socket& sock, int fd, Handler h)
{
  sendfile_op<Handler> op = { sock, fd, h, 0, 0 };
  sock.async_write_some(boost::asio::null_buffers(), op);
}
```

basic_stream_socket::native_type

(Deprecated: Use native_handle_type.) The native representation of a socket.

```
typedef StreamSocketService::native_handle_type native_type;
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::non_blocking

Gets the non-blocking mode of the socket.

```
bool non_blocking() const;
  » more...
```

Sets the non-blocking mode of the socket.

```
void non_blocking(
    bool mode);
  » more...

boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::non_blocking (1 of 3 overloads)

Inherited from basic_socket.

Gets the non-blocking mode of the socket.

```
bool non_blocking() const;
```

Return Value

`true` if the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_stream_socket::non_blocking (2 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the socket.

```
void non_blocking(
    bool mode);
```

Parameters

mode — If `true`, the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Exceptions

boost::system::system_error — Thrown on failure.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_stream_socket::non_blocking (3 of 3 overloads)

Inherited from basic_socket.

Sets the non-blocking mode of the socket.

```
boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

Parameters

mode — If `true`, the socket's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

ec — Set to indicate what error occurred, if any.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

basic_stream_socket::non_blocking_io

Inherited from socket_base.

(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.

```
typedef implementation_defined non_blocking_io;
```

Implements the FIONBIO IO control command.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::non_blocking_io command(true);
socket.io_control(command);
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::open

Open the socket using the specified protocol.

```
void open(
    const protocol_type & protocol = protocol_type());
  » more...

boost::system::error_code open(
    const protocol_type & protocol,
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::open (1 of 2 overloads)

Inherited from basic_socket.

Open the socket using the specified protocol.

```
void open(
    const protocol_type & protocol = protocol_type());
```

This function opens the socket so that it will use the specified protocol.

Parameters

protocol An object specifying protocol parameters to be used.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
socket.open(boost::asio::ip::tcp::v4());
```

basic_stream_socket::open (2 of 2 overloads)

Inherited from basic_socket.

Open the socket using the specified protocol.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code open(
    const protocol_type & protocol,
    boost::system::error_code & ec);
```

This function opens the socket so that it will use the specified protocol.

Parameters

protocol — An object specifying which protocol is to be used.

ec — Set to indicate what error occurred, if any.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
boost::system::error_code ec;
socket.open(boost::asio::ip::tcp::v4(), ec);
if (ec)
{
  // An error occurred.
}
```

basic_stream_socket::operator=

Move-assign a `basic_stream_socket` from another.

```
basic_stream_socket & operator=(
    basic_stream_socket && other);
```

This assignment operator moves a stream socket from one object to another.

Parameters

other — The other `basic_stream_socket` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_stream_socket(io_service&)` `constructor`.

basic_stream_socket::protocol_type

The protocol type.

```
typedef Protocol protocol_type;
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::read_some

Read some data from the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
  » more...

template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::read_some (1 of 2 overloads)

Read some data from the socket.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
```

This function is used to read data from the stream socket. The function call will block until one or more bytes of data has been read successfully, or until an error occurs.

Parameters

buffers — One or more buffers into which the data will be read.

Return Value

The number of bytes read.

Exceptions

boost::system::system_error — Thrown on failure. An error code of `boost::asio::error::eof` indicates that the connection was closed by the peer.

Remarks

The read_some operation may not read all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

Example

To read into a single data buffer use the `buffer` function as follows:

```
socket.read_some(boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_stream_socket::read_some (2 of 2 overloads)

Read some data from the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to read data from the stream socket. The function call will block until one or more bytes of data has been read successfully, or until an error occurs.

Parameters

buffers One or more buffers into which the data will be read.

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes read. Returns 0 if an error occurred.

Remarks

The read_some operation may not read all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

basic_stream_socket::receive

Receive some data on the socket.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers);
  » more...

template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags);
  » more...
```

Receive some data on a connected socket.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::receive (1 of 3 overloads)

Receive some data on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers);
```

This function is used to receive data on the stream socket. The function call will block until one or more bytes of data has been received successfully, or until an error occurs.

Parameters

buffers One or more buffers into which the data will be received.

Return Value

The number of bytes received.

Exceptions

boost::system::system_error Thrown on failure. An error code of `boost::asio::error::eof` indicates that the connection was closed by the peer.

Remarks

The receive operation may not receive all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

Example

To receive into a single data buffer use the `buffer` function as follows:

```
socket.receive(boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_stream_socket::receive (2 of 3 overloads)

Receive some data on the socket.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags);
```

This function is used to receive data on the stream socket. The function call will block until one or more bytes of data has been received successfully, or until an error occurs.

Parameters

buffers One or more buffers into which the data will be received.

flags Flags specifying how the receive call is to be made.

Return Value

The number of bytes received.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure. An error code of `boost::asio::error::eof` indicates that the connection was closed by the peer.

Remarks

The receive operation may not receive all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

Example

To receive into a single data buffer use the `buffer` function as follows:

```
socket.receive(boost::asio::buffer(data, size), 0);
```

See the `buffer` documentation for information on receiving into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_stream_socket::receive (3 of 3 overloads)

Receive some data on a connected socket.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

This function is used to receive data on the stream socket. The function call will block until one or more bytes of data has been received successfully, or until an error occurs.

Parameters

buffers	One or more buffers into which the data will be received.
flags	Flags specifying how the receive call is to be made.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes received. Returns 0 if an error occurred.

Remarks

The receive operation may not receive all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

basic_stream_socket::receive_buffer_size

Inherited from socket_base.

Socket option for the receive buffer size of a socket.

```
typedef implementation_defined receive_buffer_size;
```

Implements the SOL_SOCKET/SO_RCVBUF socket option.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::receive_low_watermark

Inherited from socket_base.

Socket option for the receive low watermark.

```
typedef implementation_defined receive_low_watermark;
```

Implements the SOL_SOCKET/SO_RCVLOWAT socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_stream_socket::remote_endpoint

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint() const;
  » more...

endpoint_type remote_endpoint(
    boost::system::error_code & ec) const;
  » more...
```

basic_stream_socket::remote_endpoint (1 of 2 overloads)

Inherited from basic_socket.

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint() const;
```

This function is used to obtain the remote endpoint of the socket.

Return Value

An object that represents the remote endpoint of the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::endpoint endpoint = socket.remote_endpoint();
```

basic_stream_socket::remote_endpoint (2 of 2 overloads)

Inherited from basic_socket.

Get the remote endpoint of the socket.

```
endpoint_type remote_endpoint(
    boost::system::error_code & ec) const;
```

This function is used to obtain the remote endpoint of the socket.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

An object that represents the remote endpoint of the socket. Returns a default-constructed endpoint object if an error occurred.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
boost::asio::ip::tcp::endpoint endpoint = socket.remote_endpoint(ec);
if (ec)
{
  // An error occurred.
}
```

basic_stream_socket::reuse_address

Inherited from socket_base.

Socket option to allow the socket to be bound to an address that is already in use.

```
typedef implementation_defined reuse_address;
```

Implements the SOL_SOCKET/SO_REUSEADDR socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option(true);
acceptor.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::send

Send some data on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers);
  » more...

template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags);
  » more...

template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::send (1 of 3 overloads)

Send some data on the socket.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers);
```

This function is used to send data on the stream socket. The function call will block until one or more bytes of the data has been sent successfully, or an until error occurs.

Parameters

buffers One or more data buffers to be sent on the socket.

Return Value

The number of bytes sent.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

The send operation may not transmit all of the data to the peer. Consider using the `write` function if you need to ensure that all data is written before the blocking operation completes.

Example

To send a single data buffer use the `buffer` function as follows:

```
socket.send(boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on sending multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_stream_socket::send (2 of 3 overloads)

Send some data on the socket.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags);
```

This function is used to send data on the stream socket. The function call will block until one or more bytes of the data has been sent successfully, or an until error occurs.

Parameters

buffers One or more data buffers to be sent on the socket.

flags Flags specifying how the send call is to be made.

Return Value

The number of bytes sent.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

The send operation may not transmit all of the data to the peer. Consider using the `write` function if you need to ensure that all data is written before the blocking operation completes.

Example

To send a single data buffer use the `buffer` function as follows:

```
socket.send(boost::asio::buffer(data, size), 0);
```

See the `buffer` documentation for information on sending multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_stream_socket::send (3 of 3 overloads)

Send some data on the socket.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

This function is used to send data on the stream socket. The function call will block until one or more bytes of the data has been sent successfully, or an until error occurs.

Parameters

buffers One or more data buffers to be sent on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

flags Flags specifying how the send call is to be made.

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes sent. Returns 0 if an error occurred.

Remarks

The send operation may not transmit all of the data to the peer. Consider using the `write` function if you need to ensure that all data is written before the blocking operation completes.

basic_stream_socket::send_buffer_size

Inherited from socket_base.

Socket option for the send buffer size of a socket.

```
typedef implementation_defined send_buffer_size;
```

Implements the SOL_SOCKET/SO_SNDBUF socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::send_low_watermark

Inherited from socket_base.

Socket option for the send low watermark.

```
typedef implementation_defined send_low_watermark;
```

Implements the SOL_SOCKET/SO_SNDLOWAT socket option.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

basic_stream_socket::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef StreamSocketService service_type;
```

Requirements

Header: `boost/asio/basic_stream_socket.hpp`

Convenience header: `boost/asio.hpp`

basic_stream_socket::set_option

Set an option on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void set_option(
    const SettableSocketOption & option);
  » more...

boost::system::error_code set_option(
    const SettableSocketOption & option,
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::set_option (1 of 2 overloads)

Inherited from basic_socket.

Set an option on the socket.

```
template<
    typename SettableSocketOption>
void set_option(
    const SettableSocketOption & option);
```

This function is used to set an option on the socket.

Parameters

option The new option value to be set on the socket.

Exceptions

boost::system::system_error Thrown on failure.

Example

Setting the IPPROTO_TCP/TCP_NODELAY option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::no_delay option(true);
socket.set_option(option);
```

basic_stream_socket::set_option (2 of 2 overloads)

Inherited from basic_socket.

Set an option on the socket.

```
template<
    typename SettableSocketOption>
boost::system::error_code set_option(
    const SettableSocketOption & option,
    boost::system::error_code & ec);
```

This function is used to set an option on the socket.

Parameters

option The new option value to be set on the socket.

ec Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

Setting the IPPROTO_TCP/TCP_NODELAY option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::no_delay option(true);
boost::system::error_code ec;
socket.set_option(option, ec);
if (ec)
{
  // An error occurred.
}
```

basic_stream_socket::shutdown

Disable sends or receives on the socket.

```
void shutdown(
    shutdown_type what);
  » more...

boost::system::error_code shutdown(
    shutdown_type what,
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::shutdown (1 of 2 overloads)

Inherited from basic_socket.

Disable sends or receives on the socket.

```
void shutdown(
    shutdown_type what);
```

This function is used to disable send operations, receive operations, or both.

Parameters

what Determines what types of operation will no longer be allowed.

Exceptions

boost::system::system_error Thrown on failure.

Example

Shutting down the send side of the socket:

```
boost::asio::ip::tcp::socket socket(io_service);
...
socket.shutdown(boost::asio::ip::tcp::socket::shutdown_send);
```

basic_stream_socket::shutdown (2 of 2 overloads)

Inherited from basic_socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Disable sends or receives on the socket.

```
boost::system::error_code shutdown(
    shutdown_type what,
    boost::system::error_code & ec);
```

This function is used to disable send operations, receive operations, or both.

Parameters

what Determines what types of operation will no longer be allowed.

ec Set to indicate what error occurred, if any.

Example

Shutting down the send side of the socket:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::system::error_code ec;
socket.shutdown(boost::asio::ip::tcp::socket::shutdown_send, ec);
if (ec)
{
  // An error occurred.
}
```

basic_stream_socket::shutdown_type

Inherited from socket_base.

Different ways a socket may be shutdown.

```
enum shutdown_type
```

Values

shutdown_receive Shutdown the receive side of the socket.

shutdown_send Shutdown the send side of the socket.

shutdown_both Shutdown both send and receive on the socket.

basic_stream_socket::write_some

Write some data to the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
  » more...

template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

basic_stream_socket::write_some (1 of 2 overloads)

Write some data to the socket.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
```

This function is used to write data to the stream socket. The function call will block until one or more bytes of the data has been written successfully, or until an error occurs.

Parameters

buffers — One or more data buffers to be written to the socket.

Return Value

The number of bytes written.

Exceptions

boost::system::system_error — Thrown on failure. An error code of `boost::asio::error::eof` indicates that the connection was closed by the peer.

Remarks

The write_some operation may not transmit all of the data to the peer. Consider using the `write` function if you need to ensure that all data is written before the blocking operation completes.

Example

To write a single data buffer use the `buffer` function as follows:

```
socket.write_some(boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

basic_stream_socket::write_some (2 of 2 overloads)

Write some data to the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to write data to the stream socket. The function call will block until one or more bytes of the data has been written successfully, or until an error occurs.

Parameters

buffers One or more data buffers to be written to the socket.

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes written. Returns 0 if an error occurred.

Remarks

The write_some operation may not transmit all of the data to the peer. Consider using the `write` function if you need to ensure that all data is written before the blocking operation completes.

basic_streambuf

Automatically resizable buffer class based on std::streambuf.

```
template<
    typename Allocator = std::allocator<char>>
class basic_streambuf :
  noncopyable
```

Types

Name	Description
const_buffers_type	The type used to represent the input sequence as a list of buffers.
mutable_buffers_type	The type used to represent the output sequence as a list of buffers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
basic_streambuf	Construct a basic_streambuf object.
commit	Move characters from the output sequence to the input sequence.
consume	Remove characters from the input sequence.
data	Get a list of buffers that represents the input sequence.
max_size	Get the maximum size of the basic_streambuf.
prepare	Get a list of buffers that represents the output sequence, with the given size.
size	Get the size of the input sequence.

Protected Member Functions

Name	Description
overflow	Override std::streambuf behaviour.
reserve	
underflow	Override std::streambuf behaviour.

The `basic_streambuf` class is derived from `std::streambuf` to associate the streambuf's input and output sequences with one or more character arrays. These character arrays are internal to the `basic_streambuf` object, but direct access to the array elements is provided to permit them to be used efficiently with I/O operations. Characters written to the output sequence of a `basic_streambuf` object are appended to the input sequence of the same object.

The `basic_streambuf` class's public interface is intended to permit the following implementation strategies:

- A single contiguous character array, which is reallocated as necessary to accommodate changes in the size of the character sequence. This is the implementation approach currently used in Asio.
- A sequence of one or more character arrays, where each array is of the same size. Additional character array objects are appended to the sequence to accommodate changes in the size of the character sequence.
- A sequence of one or more character arrays of varying sizes. Additional character array objects are appended to the sequence to accommodate changes in the size of the character sequence.

The constructor for `basic_streambuf` accepts a `size_t` argument specifying the maximum of the sum of the sizes of the input sequence and output sequence. During the lifetime of the `basic_streambuf` object, the following invariant holds:

```
size() <= max_size()
```

Any member function that would, if successful, cause the invariant to be violated shall throw an exception of class `std::length_error`.

The constructor for `basic_streambuf` takes an Allocator argument. A copy of this argument is used for any memory allocation performed, by the constructor and by all member functions, during the lifetime of each `basic_streambuf` object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Examples

Writing directly from an streambuf to a socket:

```
boost::asio::streambuf b;
std::ostream os(&b);
os << "Hello, World!\n";

// try sending some data in input sequence
size_t n = sock.send(b.data());

b.consume(n); // sent data is removed from input sequence
```

Reading from a socket directly into a streambuf:

```
boost::asio::streambuf b;

// reserve 512 bytes in output sequence
boost::asio::streambuf::mutable_buffers_type bufs = b.prepare(512);

size_t n = sock.receive(bufs);

// received data is "committed" from output sequence to input sequence
b.commit(n);

std::istream is(&b);
std::string s;
is >> s;
```

Requirements

Header: `boost/asio/basic_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_streambuf::basic_streambuf

Construct a `basic_streambuf` object.

```
basic_streambuf(
    std::size_t maximum_size = (std::numeric_limits< std::size_t >::max)(),
    const Allocator & allocator = Allocator());
```

Constructs a streambuf with the specified maximum size. The initial size of the streambuf's input sequence is 0.

basic_streambuf::commit

Move characters from the output sequence to the input sequence.

```
void commit(
    std::size_t n);
```

Appends `n` characters from the start of the output sequence to the input sequence. The beginning of the output sequence is advanced by `n` characters.

Requires a preceding call `prepare(x)` where `x >= n`, and no intervening operations that modify the input or output sequence.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

std::length_error	If `n` is greater than the size of the output sequence.

basic_streambuf::const_buffers_type

The type used to represent the input sequence as a list of buffers.

```
typedef implementation_defined const_buffers_type;
```

Requirements

Header: `boost/asio/basic_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_streambuf::consume

Remove characters from the input sequence.

```
void consume(
    std::size_t n);
```

Removes `n` characters from the beginning of the input sequence.

Exceptions

std::length_error	If `n > size()`.

basic_streambuf::data

Get a list of buffers that represents the input sequence.

```
const_buffers_type data() const;
```

Return Value

An object of type `const_buffers_type` that satisfies ConstBufferSequence requirements, representing all character arrays in the input sequence.

Remarks

The returned object is invalidated by any `basic_streambuf` member function that modifies the input sequence or output sequence.

basic_streambuf::max_size

Get the maximum size of the `basic_streambuf`.

```
std::size_t max_size() const;
```

Return Value

The allowed maximum of the sum of the sizes of the input sequence and output sequence.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_streambuf::mutable_buffers_type

The type used to represent the output sequence as a list of buffers.

```
typedef implementation_defined mutable_buffers_type;
```

Requirements

Header: `boost/asio/basic_streambuf.hpp`

Convenience header: `boost/asio.hpp`

basic_streambuf::overflow

Override std::streambuf behaviour.

```
int_type overflow(
    int_type c);
```

Behaves according to the specification of `std::streambuf::overflow()`, with the specialisation that `std::length_error` is thrown if appending the character to the input sequence would require the condition `size() > max_size()` to be true.

basic_streambuf::prepare

Get a list of buffers that represents the output sequence, with the given size.

```
mutable_buffers_type prepare(
    std::size_t n);
```

Ensures that the output sequence can accommodate `n` characters, reallocating character array objects as necessary.

Return Value

An object of type `mutable_buffers_type` that satisfies MutableBufferSequence requirements, representing character array objects at the start of the output sequence such that the sum of the buffer sizes is `n`.

Exceptions

std::length_error	If `size() + n > max_size()`.

Remarks

The returned object is invalidated by any `basic_streambuf` member function that modifies the input sequence or output sequence.

basic_streambuf::reserve

```
void reserve(
    std::size_t n);
```

basic_streambuf::size

Get the size of the input sequence.

```
std::size_t size() const;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Return Value

The size of the input sequence. The value is equal to that calculated for `s` in the following code:

```
size_t s = 0;
const_buffers_type bufs = data();
const_buffers_type::const_iterator i = bufs.begin();
while (i != bufs.end())
{
  const_buffer buf(*i++);
  s += buffer_size(buf);
}
```

basic_streambuf::underflow

Override std::streambuf behaviour.

```
int_type underflow();
```

Behaves according to the specification of `std::streambuf::underflow()`.

basic_waitable_timer

Provides waitable timer functionality.

```
template<
    typename Clock,
    typename WaitTraits = boost::asio::wait_traits<Clock>,
    typename WaitableTimerService = waitable_timer_service<Clock, WaitTraits>>
class basic_waitable_timer :
  public basic_io_object< WaitableTimerService >
```

Types

Name	Description
clock_type	The clock type.
duration	The duration type of the clock.
implementation_type	The underlying implementation type of I/O object.
service_type	The type of the service that will be used to provide I/O operations.
time_point	The time point type of the clock.
traits_type	The wait traits type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
async_wait	Start an asynchronous wait on the timer.
basic_waitable_timer	Constructor. Constructor to set a particular expiry time as an absolute time. Constructor to set a particular expiry time relative to now.
cancel	Cancel any asynchronous operations that are waiting on the timer.
cancel_one	Cancels one asynchronous operation that is waiting on the timer.
expires_at	Get the timer's expiry time as an absolute time. Set the timer's expiry time as an absolute time.
expires_from_now	Get the timer's expiry time relative to now. Set the timer's expiry time relative to now.
get_io_service	Get the io_service associated with the object.
wait	Perform a blocking wait on the timer.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_waitable_timer` class template provides the ability to perform a blocking or asynchronous wait for a timer to expire.

A waitable timer is always in one of two states: "expired" or "not expired". If the `wait()` or `async_wait()` function is called on an expired timer, the wait operation will complete immediately.

Most applications will use the boost::asio::waitable_timer typedef.

Remarks

This waitable timer functionality is for use with the C++11 standard library's `<chrono>` facility, or with the Boost.Chrono library.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Examples

Performing a blocking wait:

```
// Construct a timer without setting an expiry time.
boost::asio::waitable_timer timer(io_service);

// Set an expiry time relative to now.
timer.expires_from_now(boost::posix_time::seconds(5));

// Wait for the timer to expire.
timer.wait();
```

Performing an asynchronous wait:

```
void handler(const boost::system::error_code& error)
{
  if (!error)
  {
    // Timer expired.
  }
}

...

// Construct a timer with an absolute expiry time.
boost::asio::waitable_timer timer(io_service,
    boost::posix_time::time_from_string("2005-12-07 23:59:59.000"));

// Start an asynchronous wait.
timer.async_wait(handler);
```

Changing an active waitable_timer's expiry time

Changing the expiry time of a timer while there are pending asynchronous waits causes those wait operations to be cancelled. To ensure that the action associated with the timer is performed only once, use something like this: used:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void on_some_event()
{
  if (my_timer.expires_from_now(seconds(5)) > 0)
  {
    // We managed to cancel the timer. Start new asynchronous wait.
    my_timer.async_wait(on_timeout);
  }
  else
  {
    // Too late, timer has already expired!
  }
}

void on_timeout(const boost::system::error_code& e)
{
  if (e != boost::asio::error::operation_aborted)
  {
    // Timer was not cancelled, take necessary action.
  }
}
```

- The `boost::asio::basic_waitable_timer::expires_from_now()` function cancels any pending asynchronous waits, and returns the number of asynchronous waits that were cancelled. If it returns 0 then you were too late and the wait handler has already been executed, or will soon be executed. If it returns 1 then the wait handler was successfully cancelled.
- If a wait handler is cancelled, the boost::system::error_code passed to it contains the value `boost::asio::error::operation_aborted`.

Requirements

Header: `boost/asio/basic_waitable_timer.hpp`

Convenience header: `boost/asio.hpp`

basic_waitable_timer::async_wait

Start an asynchronous wait on the timer.

```
template<
    typename WaitHandler>
void async_wait(
    WaitHandler handler);
```

This function may be used to initiate an asynchronous wait against the timer. It always returns immediately.

For each call to `async_wait()`, the supplied handler will be called exactly once. The handler will be called when:

- The timer has expired.
- The timer was cancelled, in which case the handler is passed the error code `boost::asio::error::operation_aborted`.

Parameters

handler — The handler to be called when the timer expires. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error // Result of operation.
);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

basic_waitable_timer::basic_waitable_timer

Constructor.

```
explicit basic_waitable_timer(
    boost::asio::io_service & io_service);
  » more...
```

Constructor to set a particular expiry time as an absolute time.

```
basic_waitable_timer(
    boost::asio::io_service & io_service,
    const time_point & expiry_time);
  » more...
```

Constructor to set a particular expiry time relative to now.

```
basic_waitable_timer(
    boost::asio::io_service & io_service,
    const duration & expiry_time);
  » more...
```

basic_waitable_timer::basic_waitable_timer (1 of 3 overloads)

Constructor.

```
basic_waitable_timer(
    boost::asio::io_service & io_service);
```

This constructor creates a timer without setting an expiry time. The `expires_at()` or `expires_from_now()` functions must be called to set an expiry time before the timer can be waited on.

Parameters

io_service — The `io_service` object that the timer will use to dispatch handlers for any asynchronous operations performed on the timer.

basic_waitable_timer::basic_waitable_timer (2 of 3 overloads)

Constructor to set a particular expiry time as an absolute time.

```
basic_waitable_timer(
    boost::asio::io_service & io_service,
    const time_point & expiry_time);
```

This constructor creates a timer and sets the expiry time.

Parameters

io_service — The `io_service` object that the timer will use to dispatch handlers for any asynchronous operations performed on the timer.

expiry_time — The expiry time to be used for the timer, expressed as an absolute time.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_waitable_timer::basic_waitable_timer (3 of 3 overloads)

Constructor to set a particular expiry time relative to now.

```
basic_waitable_timer(
    boost::asio::io_service & io_service,
    const duration & expiry_time);
```

This constructor creates a timer and sets the expiry time.

Parameters

io_service	The `io_service` object that the timer will use to dispatch handlers for any asynchronous operations performed on the timer.
expiry_time	The expiry time to be used for the timer, relative to now.

basic_waitable_timer::cancel

Cancel any asynchronous operations that are waiting on the timer.

```
std::size_t cancel();
  » more...

std::size_t cancel(
    boost::system::error_code & ec);
  » more...
```

basic_waitable_timer::cancel (1 of 2 overloads)

Cancel any asynchronous operations that are waiting on the timer.

```
std::size_t cancel();
```

This function forces the completion of any pending asynchronous wait operations against the timer. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Cancelling the timer does not change the expiry time.

Return Value

The number of asynchronous operations that were cancelled.

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

If the timer has already expired when `cancel()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_waitable_timer::cancel (2 of 2 overloads)

Cancel any asynchronous operations that are waiting on the timer.

```
std::size_t cancel(
    boost::system::error_code & ec);
```

This function forces the completion of any pending asynchronous wait operations against the timer. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Cancelling the timer does not change the expiry time.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

The number of asynchronous operations that were cancelled.

Remarks

If the timer has already expired when `cancel()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_waitable_timer::cancel_one

Cancels one asynchronous operation that is waiting on the timer.

```
std::size_t cancel_one();
  » more...

std::size_t cancel_one(
    boost::system::error_code & ec);
  » more...
```

basic_waitable_timer::cancel_one (1 of 2 overloads)

Cancels one asynchronous operation that is waiting on the timer.

```
std::size_t cancel_one();
```

This function forces the completion of one pending asynchronous wait operation against the timer. Handlers are cancelled in FIFO order. The handler for the cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Cancelling the timer does not change the expiry time.

Return Value

The number of asynchronous operations that were cancelled. That is, either 0 or 1.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

If the timer has already expired when `cancel_one()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_waitable_timer::cancel_one (2 of 2 overloads)

Cancels one asynchronous operation that is waiting on the timer.

```
std::size_t cancel_one(
    boost::system::error_code & ec);
```

This function forces the completion of one pending asynchronous wait operation against the timer. Handlers are cancelled in FIFO order. The handler for the cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Cancelling the timer does not change the expiry time.

Parameters

ec	Set to indicate what error occurred, if any.

Return Value

The number of asynchronous operations that were cancelled. That is, either 0 or 1.

Remarks

If the timer has already expired when `cancel_one()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_waitable_timer::clock_type

The clock type.

```
typedef Clock clock_type;
```

Requirements

Header: `boost/asio/basic_waitable_timer.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_waitable_timer::duration

The duration type of the clock.

```
typedef clock_type::duration duration;
```

Requirements

Header: `boost/asio/basic_waitable_timer.hpp`

Convenience header: `boost/asio.hpp`

basic_waitable_timer::expires_at

Get the timer's expiry time as an absolute time.

```
time_point expires_at() const;
  » more...
```

Set the timer's expiry time as an absolute time.

```
std::size_t expires_at(
    const time_point & expiry_time);
  » more...

std::size_t expires_at(
    const time_point & expiry_time,
    boost::system::error_code & ec);
  » more...
```

basic_waitable_timer::expires_at (1 of 3 overloads)

Get the timer's expiry time as an absolute time.

```
time_point expires_at() const;
```

This function may be used to obtain the timer's current expiry time. Whether the timer has expired or not does not affect this value.

basic_waitable_timer::expires_at (2 of 3 overloads)

Set the timer's expiry time as an absolute time.

```
std::size_t expires_at(
    const time_point & expiry_time);
```

This function sets the expiry time. Any pending asynchronous wait operations will be cancelled. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Parameters

expiry_time	The expiry time to be used for the timer.

Return Value

The number of asynchronous operations that were cancelled.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

If the timer has already expired when `expires_at()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_waitable_timer::expires_at (3 of 3 overloads)

Set the timer's expiry time as an absolute time.

```
std::size_t expires_at(
    const time_point & expiry_time,
    boost::system::error_code & ec);
```

This function sets the expiry time. Any pending asynchronous wait operations will be cancelled. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Parameters

expiry_time	The expiry time to be used for the timer.
ec	Set to indicate what error occurred, if any.

Return Value

The number of asynchronous operations that were cancelled.

Remarks

If the timer has already expired when `expires_at()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_waitable_timer::expires_from_now

Get the timer's expiry time relative to now.

```
duration expires_from_now() const;
  » more...
```

Set the timer's expiry time relative to now.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t expires_from_now(
    const duration & expiry_time);
  » more...

std::size_t expires_from_now(
    const duration & expiry_time,
    boost::system::error_code & ec);
  » more...
```

basic_waitable_timer::expires_from_now (1 of 3 overloads)

Get the timer's expiry time relative to now.

```
duration expires_from_now() const;
```

This function may be used to obtain the timer's current expiry time. Whether the timer has expired or not does not affect this value.

basic_waitable_timer::expires_from_now (2 of 3 overloads)

Set the timer's expiry time relative to now.

```
std::size_t expires_from_now(
    const duration & expiry_time);
```

This function sets the expiry time. Any pending asynchronous wait operations will be cancelled. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Parameters

expiry_time The expiry time to be used for the timer.

Return Value

The number of asynchronous operations that were cancelled.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

If the timer has already expired when `expires_from_now()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_waitable_timer::expires_from_now (3 of 3 overloads)

Set the timer's expiry time relative to now.

```
std::size_t expires_from_now(
    const duration & expiry_time,
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This function sets the expiry time. Any pending asynchronous wait operations will be cancelled. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

Parameters

expiry_time	The expiry time to be used for the timer.
ec	Set to indicate what error occurred, if any.

Return Value

The number of asynchronous operations that were cancelled.

Remarks

If the timer has already expired when `expires_from_now()` is called, then the handlers for asynchronous wait operations will:

- have already been invoked; or
- have been queued for invocation in the near future.

These handlers can no longer be cancelled, and therefore are passed an error code that indicates the successful completion of the wait operation.

basic_waitable_timer::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

basic_waitable_timer::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

basic_waitable_timer::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

basic_waitable_timer::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

basic_waitable_timer::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

basic_waitable_timer::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

basic_waitable_timer::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

basic_waitable_timer::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

basic_waitable_timer::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/basic_waitable_timer.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_waitable_timer::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

basic_waitable_timer::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef WaitableTimerService service_type;
```

Requirements

Header: `boost/asio/basic_waitable_timer.hpp`

Convenience header: `boost/asio.hpp`

basic_waitable_timer::time_point

The time point type of the clock.

```
typedef clock_type::time_point time_point;
```

Requirements

Header: `boost/asio/basic_waitable_timer.hpp`

Convenience header: `boost/asio.hpp`

basic_waitable_timer::traits_type

The wait traits type.

```
typedef WaitTraits traits_type;
```

Requirements

Header: `boost/asio/basic_waitable_timer.hpp`

Convenience header: `boost/asio.hpp`

basic_waitable_timer::wait

Perform a blocking wait on the timer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void wait();
  » more...

void wait(
    boost::system::error_code & ec);
  » more...
```

basic_waitable_timer::wait (1 of 2 overloads)

Perform a blocking wait on the timer.

```
void wait();
```

This function is used to wait for the timer to expire. This function blocks and does not return until the timer has expired.

Exceptions

boost::system::system_error	Thrown on failure.

basic_waitable_timer::wait (2 of 2 overloads)

Perform a blocking wait on the timer.

```
void wait(
    boost::system::error_code & ec);
```

This function is used to wait for the timer to expire. This function blocks and does not return until the timer has expired.

Parameters

ec	Set to indicate what error occurred, if any.

buffer

The `boost::asio::buffer` function is used to create a buffer object to represent raw memory, an array of POD elements, a vector of POD elements, or a std::string.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
mutable_buffers_1 buffer(
    const mutable_buffer & b);
  » more...

mutable_buffers_1 buffer(
    const mutable_buffer & b,
    std::size_t max_size_in_bytes);
  » more...

const_buffers_1 buffer(
    const const_buffer & b);
  » more...

const_buffers_1 buffer(
    const const_buffer & b,
    std::size_t max_size_in_bytes);
  » more...

mutable_buffers_1 buffer(
    void * data,
    std::size_t size_in_bytes);
  » more...

const_buffers_1 buffer(
    const void * data,
    std::size_t size_in_bytes);
  » more...

template<
    typename PodType,
    std::size_t N>
mutable_buffers_1 buffer(
    PodType (&data)[N]);
  » more...

template<
    typename PodType,
    std::size_t N>
mutable_buffers_1 buffer(
    PodType (&data)[N],
    std::size_t max_size_in_bytes);
  » more...

template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    const PodType (&data)[N]);
  » more...

template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    const PodType (&data)[N],
    std::size_t max_size_in_bytes);
  » more...

template<
    typename PodType,
    std::size_t N>
mutable_buffers_1 buffer(
    boost::array< PodType, N > & data);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  » more...

template<
    typename PodType,
    std::size_t N>
mutable_buffers_1 buffer(
    boost::array< PodType, N > & data,
    std::size_t max_size_in_bytes);
  » more...

template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    boost::array< const PodType, N > & data);
  » more...

template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    boost::array< const PodType, N > & data,
    std::size_t max_size_in_bytes);
  » more...

template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    const boost::array< PodType, N > & data);
  » more...

template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    const boost::array< PodType, N > & data,
    std::size_t max_size_in_bytes);
  » more...

template<
    typename PodType,
    std::size_t N>
mutable_buffers_1 buffer(
    std::array< PodType, N > & data);
  » more...

template<
    typename PodType,
    std::size_t N>
mutable_buffers_1 buffer(
    std::array< PodType, N > & data,
    std::size_t max_size_in_bytes);
  » more...

template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    std::array< const PodType, N > & data);
  » more...

template<
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    std::array< const PodType, N > & data,
    std::size_t max_size_in_bytes);
  » more...

template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    const std::array< PodType, N > & data);
  » more...

template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    const std::array< PodType, N > & data,
    std::size_t max_size_in_bytes);
  » more...

template<
    typename PodType,
    typename Allocator>
mutable_buffers_1 buffer(
    std::vector< PodType, Allocator > & data);
  » more...

template<
    typename PodType,
    typename Allocator>
mutable_buffers_1 buffer(
    std::vector< PodType, Allocator > & data,
    std::size_t max_size_in_bytes);
  » more...

template<
    typename PodType,
    typename Allocator>
const_buffers_1 buffer(
    const std::vector< PodType, Allocator > & data);
  » more...

template<
    typename PodType,
    typename Allocator>
const_buffers_1 buffer(
    const std::vector< PodType, Allocator > & data,
    std::size_t max_size_in_bytes);
  » more...

template<
    typename Elem,
    typename Traits,
    typename Allocator>
const_buffers_1 buffer(
    const std::basic_string< Elem, Traits, Allocator > & data);
  » more...

template<
    typename Elem,
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    typename Traits,
    typename Allocator>
const_buffers_1 buffer(
    const std::basic_string< Elem, Traits, Allocator > & data,
    std::size_t max_size_in_bytes);
  » more...
```

A buffer object represents a contiguous region of memory as a 2-tuple consisting of a pointer and size in bytes. A tuple of the form `{void*, size_t}` specifies a mutable (modifiable) region of memory. Similarly, a tuple of the form `{const void*, size_t}` specifies a const (non-modifiable) region of memory. These two forms correspond to the classes `mutable_buffer` and `const_buffer`, respectively. To mirror C++'s conversion rules, a `mutable_buffer` is implicitly convertible to a `const_buffer`, and the opposite conversion is not permitted.

The simplest use case involves reading or writing a single buffer of a specified size:

```
sock.send(boost::asio::buffer(data, size));
```

In the above example, the return value of `boost::asio::buffer` meets the requirements of the ConstBufferSequence concept so that it may be directly passed to the socket's write function. A buffer created for modifiable memory also meets the requirements of the MutableBufferSequence concept.

An individual buffer may be created from a builtin array, std::vector, std::array or boost::array of POD elements. This helps prevent buffer overruns by automatically determining the size of the buffer:

```
char d1[128];
size_t bytes_transferred = sock.receive(boost::asio::buffer(d1));

std::vector<char> d2(128);
bytes_transferred = sock.receive(boost::asio::buffer(d2));

std::array<char, 128> d3;
bytes_transferred = sock.receive(boost::asio::buffer(d3));

boost::array<char, 128> d4;
bytes_transferred = sock.receive(boost::asio::buffer(d4));
```

In all three cases above, the buffers created are exactly 128 bytes long. Note that a vector is **never** automatically resized when creating or using a buffer. The buffer size is determined using the vector's `size()` member function, and not its capacity.

Accessing Buffer Contents

The contents of a buffer may be accessed using the `buffer_size` and `buffer_cast` functions:

```
boost::asio::mutable_buffer b1 = ...;
std::size_t s1 = boost::asio::buffer_size(b1);
unsigned char* p1 = boost::asio::buffer_cast<unsigned char*>(b1);

boost::asio::const_buffer b2 = ...;
std::size_t s2 = boost::asio::buffer_size(b2);
const void* p2 = boost::asio::buffer_cast<const void*>(b2);
```

The `boost::asio::buffer_cast` function permits violations of type safety, so uses of it in application code should be carefully considered.

For convenience, the `buffer_size` function also works on buffer sequences (that is, types meeting the ConstBufferSequence or MutableBufferSequence type requirements). In this case, the function returns the total size of all buffers in the sequence.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Buffer Copying

The `buffer_copy` function may be used to copy raw bytes between individual buffers and buffer sequences.

In particular, when used with the `buffer_size` , the `buffer_copy` function can be used to linearise a sequence of buffers. For example:

```
vector<const_buffer> buffers = ...;

vector<unsigned char> data(boost::asio::buffer_size(buffers));
boost::asio::buffer_copy(boost::asio::buffer(data), buffers);
```

Buffer Invalidation

A buffer object does not have any ownership of the memory it refers to. It is the responsibility of the application to ensure the memory region remains valid until it is no longer required for an I/O operation. When the memory is no longer available, the buffer is said to have been invalidated.

For the `boost::asio::buffer` overloads that accept an argument of type std::vector, the buffer objects returned are invalidated by any vector operation that also invalidates all references, pointers and iterators referring to the elements in the sequence (C++ Std, 23.2.4)

For the `boost::asio::buffer` overloads that accept an argument of type std::basic_string, the buffer objects returned are invalidated according to the rules defined for invalidation of references, pointers and iterators referring to elements of the sequence (C++ Std, 21.3).

Buffer Arithmetic

Buffer objects may be manipulated using simple arithmetic in a safe way which helps prevent buffer overruns. Consider an array initialised as follows:

```
boost::array<char, 6> a = { 'a', 'b', 'c', 'd', 'e' };
```

A buffer object `b1` created using:

```
b1 = boost::asio::buffer(a);
```

represents the entire array, `{ 'a', 'b', 'c', 'd', 'e' }`. An optional second argument to the `boost::asio::buffer` function may be used to limit the size, in bytes, of the buffer:

```
b2 = boost::asio::buffer(a, 3);
```

such that `b2` represents the data `{ 'a', 'b', 'c' }`. Even if the size argument exceeds the actual size of the array, the size of the buffer object created will be limited to the array size.

An offset may be applied to an existing buffer to create a new one:

```
b3 = b1 + 2;
```

where `b3` will set to represent `{ 'c', 'd', 'e' }`. If the offset exceeds the size of the existing buffer, the newly created buffer will be empty.

Both an offset and size may be specified to create a buffer that corresponds to a specific range of bytes within an existing buffer:

```
b4 = boost::asio::buffer(b1 + 1, 3);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

so that `b4` will refer to the bytes `{ 'b', 'c', 'd' }`.

Buffers and Scatter-Gather I/O

To read or write using multiple buffers (i.e. scatter-gather I/O), multiple buffer objects may be assigned into a container that supports the MutableBufferSequence (for read) or ConstBufferSequence (for write) concepts:

```
char d1[128];
std::vector<char> d2(128);
boost::array<char, 128> d3;

boost::array<mutable_buffer, 3> bufs1 = {
  boost::asio::buffer(d1),
  boost::asio::buffer(d2),
  boost::asio::buffer(d3) };
bytes_transferred = sock.receive(bufs1);

std::vector<const_buffer> bufs2;
bufs2.push_back(boost::asio::buffer(d1));
bufs2.push_back(boost::asio::buffer(d2));
bufs2.push_back(boost::asio::buffer(d3));
bytes_transferred = sock.send(bufs2);
```

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

buffer (1 of 28 overloads)

Create a new modifiable buffer from an existing buffer.

```
mutable_buffers_1 buffer(
    const mutable_buffer & b);
```

Return Value

`mutable_buffers_1(b)`.

buffer (2 of 28 overloads)

Create a new modifiable buffer from an existing buffer.

```
mutable_buffers_1 buffer(
    const mutable_buffer & b,
    std::size_t max_size_in_bytes);
```

Return Value

A `mutable_buffers_1` value equivalent to:

```
mutable_buffers_1(
    buffer_cast<void*>(b),
    min(buffer_size(b), max_size_in_bytes));
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffer (3 of 28 overloads)

Create a new non-modifiable buffer from an existing buffer.

```
const_buffers_1 buffer(
    const const_buffer & b);
```

Return Value

`const_buffers_1(b)`.

buffer (4 of 28 overloads)

Create a new non-modifiable buffer from an existing buffer.

```
const_buffers_1 buffer(
    const const_buffer & b,
    std::size_t max_size_in_bytes);
```

Return Value

A `const_buffers_1` value equivalent to:

```
const_buffers_1(
    buffer_cast<const void*>(b),
    min(buffer_size(b), max_size_in_bytes));
```

buffer (5 of 28 overloads)

Create a new modifiable buffer that represents the given memory range.

```
mutable_buffers_1 buffer(
    void * data,
    std::size_t size_in_bytes);
```

Return Value

`mutable_buffers_1(data, size_in_bytes)`.

buffer (6 of 28 overloads)

Create a new non-modifiable buffer that represents the given memory range.

```
const_buffers_1 buffer(
    const void * data,
    std::size_t size_in_bytes);
```

Return Value

`const_buffers_1(data, size_in_bytes)`.

buffer (7 of 28 overloads)

Create a new modifiable buffer that represents the given POD array.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename PodType,
    std::size_t N>
mutable_buffers_1 buffer(
    PodType (&data)[N]);
```

Return Value

A `mutable_buffers_1` value equivalent to:

```
mutable_buffers_1(
    static_cast<void*>(data),
    N * sizeof(PodType));
```

buffer (8 of 28 overloads)

Create a new modifiable buffer that represents the given POD array.

```
template<
    typename PodType,
    std::size_t N>
mutable_buffers_1 buffer(
    PodType (&data)[N],
    std::size_t max_size_in_bytes);
```

Return Value

A `mutable_buffers_1` value equivalent to:

```
mutable_buffers_1(
    static_cast<void*>(data),
    min(N * sizeof(PodType), max_size_in_bytes));
```

buffer (9 of 28 overloads)

Create a new non-modifiable buffer that represents the given POD array.

```
template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    const PodType (&data)[N]);
```

Return Value

A `const_buffers_1` value equivalent to:

```
const_buffers_1(
    static_cast<const void*>(data),
    N * sizeof(PodType));
```

buffer (10 of 28 overloads)

Create a new non-modifiable buffer that represents the given POD array.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    const PodType (&data)[N],
    std::size_t max_size_in_bytes);
```

Return Value

A `const_buffers_1` value equivalent to:

```
const_buffers_1(
    static_cast<const void*>(data),
    min(N * sizeof(PodType), max_size_in_bytes));
```

buffer (11 of 28 overloads)

Create a new modifiable buffer that represents the given POD array.

```
template<
    typename PodType,
    std::size_t N>
mutable_buffers_1 buffer(
    boost::array< PodType, N > & data);
```

Return Value

A `mutable_buffers_1` value equivalent to:

```
mutable_buffers_1(
    data.data(),
    data.size() * sizeof(PodType));
```

buffer (12 of 28 overloads)

Create a new modifiable buffer that represents the given POD array.

```
template<
    typename PodType,
    std::size_t N>
mutable_buffers_1 buffer(
    boost::array< PodType, N > & data,
    std::size_t max_size_in_bytes);
```

Return Value

A `mutable_buffers_1` value equivalent to:

```
mutable_buffers_1(
    data.data(),
    min(data.size() * sizeof(PodType), max_size_in_bytes));
```

buffer (13 of 28 overloads)

Create a new non-modifiable buffer that represents the given POD array.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    boost::array< const PodType, N > & data);
```

Return Value

A `const_buffers_1` value equivalent to:

```
const_buffers_1(
    data.data(),
    data.size() * sizeof(PodType));
```

buffer (14 of 28 overloads)

Create a new non-modifiable buffer that represents the given POD array.

```
template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    boost::array< const PodType, N > & data,
    std::size_t max_size_in_bytes);
```

Return Value

A `const_buffers_1` value equivalent to:

```
const_buffers_1(
    data.data(),
    min(data.size() * sizeof(PodType), max_size_in_bytes));
```

buffer (15 of 28 overloads)

Create a new non-modifiable buffer that represents the given POD array.

```
template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    const boost::array< PodType, N > & data);
```

Return Value

A `const_buffers_1` value equivalent to:

```
const_buffers_1(
    data.data(),
    data.size() * sizeof(PodType));
```

buffer (16 of 28 overloads)

Create a new non-modifiable buffer that represents the given POD array.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    const boost::array< PodType, N > & data,
    std::size_t max_size_in_bytes);
```

Return Value

A `const_buffers_1` value equivalent to:

```
const_buffers_1(
    data.data(),
    min(data.size() * sizeof(PodType), max_size_in_bytes));
```

buffer (17 of 28 overloads)

Create a new modifiable buffer that represents the given POD array.

```
template<
    typename PodType,
    std::size_t N>
mutable_buffers_1 buffer(
    std::array< PodType, N > & data);
```

Return Value

A `mutable_buffers_1` value equivalent to:

```
mutable_buffers_1(
    data.data(),
    data.size() * sizeof(PodType));
```

buffer (18 of 28 overloads)

Create a new modifiable buffer that represents the given POD array.

```
template<
    typename PodType,
    std::size_t N>
mutable_buffers_1 buffer(
    std::array< PodType, N > & data,
    std::size_t max_size_in_bytes);
```

Return Value

A `mutable_buffers_1` value equivalent to:

```
mutable_buffers_1(
    data.data(),
    min(data.size() * sizeof(PodType), max_size_in_bytes));
```

buffer (19 of 28 overloads)

Create a new non-modifiable buffer that represents the given POD array.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    std::array< const PodType, N > & data);
```

Return Value

A `const_buffers_1` value equivalent to:

```
const_buffers_1(
    data.data(),
    data.size() * sizeof(PodType));
```

buffer (20 of 28 overloads)

Create a new non-modifiable buffer that represents the given POD array.

```
template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    std::array< const PodType, N > & data,
    std::size_t max_size_in_bytes);
```

Return Value

A `const_buffers_1` value equivalent to:

```
const_buffers_1(
    data.data(),
    min(data.size() * sizeof(PodType), max_size_in_bytes));
```

buffer (21 of 28 overloads)

Create a new non-modifiable buffer that represents the given POD array.

```
template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    const std::array< PodType, N > & data);
```

Return Value

A `const_buffers_1` value equivalent to:

```
const_buffers_1(
    data.data(),
    data.size() * sizeof(PodType));
```

buffer (22 of 28 overloads)

Create a new non-modifiable buffer that represents the given POD array.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename PodType,
    std::size_t N>
const_buffers_1 buffer(
    const std::array< PodType, N > & data,
    std::size_t max_size_in_bytes);
```

Return Value

A `const_buffers_1` value equivalent to:

```
const_buffers_1(
    data.data(),
    min(data.size() * sizeof(PodType), max_size_in_bytes));
```

buffer (23 of 28 overloads)

Create a new modifiable buffer that represents the given POD vector.

```
template<
    typename PodType,
    typename Allocator>
mutable_buffers_1 buffer(
    std::vector< PodType, Allocator > & data);
```

Return Value

A `mutable_buffers_1` value equivalent to:

```
mutable_buffers_1(
    data.size() ? &data[0] : 0,
    data.size() * sizeof(PodType));
```

Remarks

The buffer is invalidated by any vector operation that would also invalidate iterators.

buffer (24 of 28 overloads)

Create a new modifiable buffer that represents the given POD vector.

```
template<
    typename PodType,
    typename Allocator>
mutable_buffers_1 buffer(
    std::vector< PodType, Allocator > & data,
    std::size_t max_size_in_bytes);
```

Return Value

A `mutable_buffers_1` value equivalent to:

```
mutable_buffers_1(
    data.size() ? &data[0] : 0,
    min(data.size() * sizeof(PodType), max_size_in_bytes));
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

The buffer is invalidated by any vector operation that would also invalidate iterators.

buffer (25 of 28 overloads)

Create a new non-modifiable buffer that represents the given POD vector.

```
template<
    typename PodType,
    typename Allocator>
const_buffers_1 buffer(
    const std::vector< PodType, Allocator > & data);
```

Return Value

A `const_buffers_1` value equivalent to:

```
const_buffers_1(
    data.size() ? &data[0] : 0,
    data.size() * sizeof(PodType));
```

Remarks

The buffer is invalidated by any vector operation that would also invalidate iterators.

buffer (26 of 28 overloads)

Create a new non-modifiable buffer that represents the given POD vector.

```
template<
    typename PodType,
    typename Allocator>
const_buffers_1 buffer(
    const std::vector< PodType, Allocator > & data,
    std::size_t max_size_in_bytes);
```

Return Value

A `const_buffers_1` value equivalent to:

```
const_buffers_1(
    data.size() ? &data[0] : 0,
    min(data.size() * sizeof(PodType), max_size_in_bytes));
```

Remarks

The buffer is invalidated by any vector operation that would also invalidate iterators.

buffer (27 of 28 overloads)

Create a new non-modifiable buffer that represents the given string.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename Elem,
    typename Traits,
    typename Allocator>
const_buffers_1 buffer(
    const std::basic_string< Elem, Traits, Allocator > & data);
```

Return Value

`const_buffers_1(data.data(), data.size() * sizeof(Elem))`.

Remarks

The buffer is invalidated by any non-const operation called on the given string object.

buffer (28 of 28 overloads)

Create a new non-modifiable buffer that represents the given string.

```
template<
    typename Elem,
    typename Traits,
    typename Allocator>
const_buffers_1 buffer(
    const std::basic_string< Elem, Traits, Allocator > & data,
    std::size_t max_size_in_bytes);
```

Return Value

A `const_buffers_1` value equivalent to:

```
const_buffers_1(
    data.data(),
    min(data.size() * sizeof(Elem), max_size_in_bytes));
```

Remarks

The buffer is invalidated by any non-const operation called on the given string object.

buffer_cast

The `boost::asio::buffer_cast` function is used to obtain a pointer to the underlying memory region associated with a buffer.

```
template<
    typename PointerToPodType>
PointerToPodType buffer_cast(
    const mutable_buffer & b);
  » more...

template<
    typename PointerToPodType>
PointerToPodType buffer_cast(
    const const_buffer & b);
  » more...
```

Examples:

To access the memory of a non-modifiable buffer, use:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::const_buffer b1 = ...;
const unsigned char* p1 = boost::asio::buffer_cast<const unsigned char*>(b1);
```

To access the memory of a modifiable buffer, use:

```
boost::asio::mutable_buffer b2 = ...;
unsigned char* p2 = boost::asio::buffer_cast<unsigned char*>(b2);
```

The `boost::asio::buffer_cast` function permits violations of type safety, so uses of it in application code should be carefully considered.

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

buffer_cast (1 of 2 overloads)

Cast a non-modifiable buffer to a specified pointer to POD type.

```
template<
    typename PointerToPodType>
PointerToPodType buffer_cast(
    const mutable_buffer & b);
```

buffer_cast (2 of 2 overloads)

Cast a non-modifiable buffer to a specified pointer to POD type.

```
template<
    typename PointerToPodType>
PointerToPodType buffer_cast(
    const const_buffer & b);
```

buffer_copy

The `boost::asio::buffer_copy` function is used to copy bytes from a source buffer (or buffer sequence) to a target buffer (or buffer sequence).

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t buffer_copy(
    const mutable_buffer & target,
    const const_buffer & source);
  » more...

std::size_t buffer_copy(
    const mutable_buffer & target,
    const const_buffers_1 & source);
  » more...

std::size_t buffer_copy(
    const mutable_buffer & target,
    const mutable_buffer & source);
  » more...

std::size_t buffer_copy(
    const mutable_buffer & target,
    const mutable_buffers_1 & source);
  » more...

template<
    typename ConstBufferSequence>
std::size_t buffer_copy(
    const mutable_buffer & target,
    const ConstBufferSequence & source);
  » more...

std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const const_buffer & source);
  » more...

std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const const_buffers_1 & source);
  » more...

std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const mutable_buffer & source);
  » more...

std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const mutable_buffers_1 & source);
  » more...

template<
    typename ConstBufferSequence>
std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const ConstBufferSequence & source);
  » more...

template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const const_buffer & source);
  » more...

template<
    typename MutableBufferSequence>
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const const_buffers_1 & source);
  » more...

template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const mutable_buffer & source);
  » more...

template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const mutable_buffers_1 & source);
  » more...

template<
    typename MutableBufferSequence,
    typename ConstBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const ConstBufferSequence & source);
  » more...

std::size_t buffer_copy(
    const mutable_buffer & target,
    const const_buffer & source,
    std::size_t max_bytes_to_copy);
  » more...

std::size_t buffer_copy(
    const mutable_buffer & target,
    const const_buffers_1 & source,
    std::size_t max_bytes_to_copy);
  » more...

std::size_t buffer_copy(
    const mutable_buffer & target,
    const mutable_buffer & source,
    std::size_t max_bytes_to_copy);
  » more...

std::size_t buffer_copy(
    const mutable_buffer & target,
    const mutable_buffers_1 & source,
    std::size_t max_bytes_to_copy);
  » more...

template<
    typename ConstBufferSequence>
std::size_t buffer_copy(
    const mutable_buffer & target,
    const ConstBufferSequence & source,
    std::size_t max_bytes_to_copy);
  » more...

std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const const_buffer & source,
    std::size_t max_bytes_to_copy);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  » more...

std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const const_buffers_1 & source,
    std::size_t max_bytes_to_copy);
  » more...

std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const mutable_buffer & source,
    std::size_t max_bytes_to_copy);
  » more...

std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const mutable_buffers_1 & source,
    std::size_t max_bytes_to_copy);
  » more...

template<
    typename ConstBufferSequence>
std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const ConstBufferSequence & source,
    std::size_t max_bytes_to_copy);
  » more...

template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const const_buffer & source,
    std::size_t max_bytes_to_copy);
  » more...

template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const const_buffers_1 & source,
    std::size_t max_bytes_to_copy);
  » more...

template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const mutable_buffer & source,
    std::size_t max_bytes_to_copy);
  » more...

template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const mutable_buffers_1 & source,
    std::size_t max_bytes_to_copy);
  » more...

template<
    typename MutableBufferSequence,
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    typename ConstBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const ConstBufferSequence & source,
    std::size_t max_bytes_to_copy);
  » more...
```

The `buffer_copy` function is available in two forms:

- A 2-argument form: `buffer_copy(target, source)`
- A 3-argument form: `buffer_copy(target, source, max_bytes_to_copy)`

Both forms return the number of bytes actually copied. The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`
- `If` specified, `max_bytes_to_copy`.

This prevents buffer overflow, regardless of the buffer sizes used in the copy operation.

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

buffer_copy (1 of 30 overloads)

Copies bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffer & target,
    const const_buffer & source);
```

Parameters

target — A modifiable buffer representing the memory region to which the bytes will be copied.

source — A non-modifiable buffer representing the memory region from which the bytes will be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

buffer_copy (2 of 30 overloads)

Copies bytes from a source buffer to a target buffer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t buffer_copy(
    const mutable_buffer & target,
    const const_buffers_1 & source);
```

Parameters

target A modifiable buffer representing the memory region to which the bytes will be copied.

source A non-modifiable buffer representing the memory region from which the bytes will be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

buffer_copy (3 of 30 overloads)

Copies bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffer & target,
    const mutable_buffer & source);
```

Parameters

target A modifiable buffer representing the memory region to which the bytes will be copied.

source A modifiable buffer representing the memory region from which the bytes will be copied. The contents of the source buffer will not be modified.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

buffer_copy (4 of 30 overloads)

Copies bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffer & target,
    const mutable_buffers_1 & source);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

target A modifiable buffer representing the memory region to which the bytes will be copied.

source A modifiable buffer representing the memory region from which the bytes will be copied. The contents of the source buffer will not be modified.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

buffer_copy (5 of 30 overloads)

Copies bytes from a source buffer sequence to a target buffer.

```
template<
    typename ConstBufferSequence>
std::size_t buffer_copy(
    const mutable_buffer & target,
    const ConstBufferSequence & source);
```

Parameters

target A modifiable buffer representing the memory region to which the bytes will be copied.

source A non-modifiable buffer sequence representing the memory regions from which the bytes will be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

buffer_copy (6 of 30 overloads)

Copies bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const const_buffer & source);
```

Parameters

target A modifiable buffer representing the memory region to which the bytes will be copied.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

source A non-modifiable buffer representing the memory region from which the bytes will be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

buffer_copy (7 of 30 overloads)

Copies bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const const_buffers_1 & source);
```

Parameters

target A modifiable buffer representing the memory region to which the bytes will be copied.

source A non-modifiable buffer representing the memory region from which the bytes will be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

buffer_copy (8 of 30 overloads)

Copies bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const mutable_buffer & source);
```

Parameters

target A modifiable buffer representing the memory region to which the bytes will be copied.

source A modifiable buffer representing the memory region from which the bytes will be copied. The contents of the source buffer will not be modified.

Return Value

The number of bytes copied.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

buffer_copy (9 of 30 overloads)

Copies bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const mutable_buffers_1 & source);
```

Parameters

target — A modifiable buffer representing the memory region to which the bytes will be copied.

source — A modifiable buffer representing the memory region from which the bytes will be copied. The contents of the source buffer will not be modified.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

buffer_copy (10 of 30 overloads)

Copies bytes from a source buffer sequence to a target buffer.

```
template<
    typename ConstBufferSequence>
std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const ConstBufferSequence & source);
```

Parameters

target — A modifiable buffer representing the memory region to which the bytes will be copied.

source — A non-modifiable buffer sequence representing the memory regions from which the bytes will be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- `buffer_size(source)`

buffer_copy (11 of 30 overloads)

Copies bytes from a source buffer to a target buffer sequence.

```
template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const const_buffer & source);
```

Parameters

target A modifiable buffer sequence representing the memory regions to which the bytes will be copied.

source A non-modifiable buffer representing the memory region from which the bytes will be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

buffer_copy (12 of 30 overloads)

Copies bytes from a source buffer to a target buffer sequence.

```
template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const const_buffers_1 & source);
```

Parameters

target A modifiable buffer sequence representing the memory regions to which the bytes will be copied.

source A non-modifiable buffer representing the memory region from which the bytes will be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffer_copy (13 of 30 overloads)

Copies bytes from a source buffer to a target buffer sequence.

```
template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const mutable_buffer & source);
```

Parameters

target — A modifiable buffer sequence representing the memory regions to which the bytes will be copied.

source — A modifiable buffer representing the memory region from which the bytes will be copied. The contents of the source buffer will not be modified.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

buffer_copy (14 of 30 overloads)

Copies bytes from a source buffer to a target buffer sequence.

```
template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const mutable_buffers_1 & source);
```

Parameters

target — A modifiable buffer sequence representing the memory regions to which the bytes will be copied.

source — A modifiable buffer representing the memory region from which the bytes will be copied. The contents of the source buffer will not be modified.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffer_copy (15 of 30 overloads)

Copies bytes from a source buffer sequence to a target buffer sequence.

```
template<
    typename MutableBufferSequence,
    typename ConstBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const ConstBufferSequence & source);
```

Parameters

target A modifiable buffer sequence representing the memory regions to which the bytes will be copied.

source A non-modifiable buffer sequence representing the memory regions from which the bytes will be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

buffer_copy (16 of 30 overloads)

Copies a limited number of bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffer & target,
    const const_buffer & source,
    std::size_t max_bytes_to_copy);
```

Parameters

target A modifiable buffer representing the memory region to which the bytes will be copied.

source A non-modifiable buffer representing the memory region from which the bytes will be copied.

max_bytes_to_copy The maximum number of bytes to be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`
- `max_bytes_to_copy`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffer_copy (17 of 30 overloads)

Copies a limited number of bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffer & target,
    const const_buffers_1 & source,
    std::size_t max_bytes_to_copy);
```

Parameters

target	A modifiable buffer representing the memory region to which the bytes will be copied.
source	A non-modifiable buffer representing the memory region from which the bytes will be copied.
max_bytes_to_copy	The maximum number of bytes to be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`
- `max_bytes_to_copy`

buffer_copy (18 of 30 overloads)

Copies a limited number of bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffer & target,
    const mutable_buffer & source,
    std::size_t max_bytes_to_copy);
```

Parameters

target	A modifiable buffer representing the memory region to which the bytes will be copied.
source	A modifiable buffer representing the memory region from which the bytes will be copied. The contents of the source buffer will not be modified.
max_bytes_to_copy	The maximum number of bytes to be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- `max_bytes_to_copy`

buffer_copy (19 of 30 overloads)

Copies a limited number of bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffer & target,
    const mutable_buffers_1 & source,
    std::size_t max_bytes_to_copy);
```

Parameters

target	A modifiable buffer representing the memory region to which the bytes will be copied.
source	A modifiable buffer representing the memory region from which the bytes will be copied. The contents of the source buffer will not be modified.
max_bytes_to_copy	The maximum number of bytes to be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`
- `max_bytes_to_copy`

buffer_copy (20 of 30 overloads)

Copies a limited number of bytes from a source buffer sequence to a target buffer.

```
template<
    typename ConstBufferSequence>
std::size_t buffer_copy(
    const mutable_buffer & target,
    const ConstBufferSequence & source,
    std::size_t max_bytes_to_copy);
```

Parameters

target	A modifiable buffer representing the memory region to which the bytes will be copied.
source	A non-modifiable buffer sequence representing the memory regions from which the bytes will be copied.
max_bytes_to_copy	The maximum number of bytes to be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- `buffer_size(target)`
- `buffer_size(source)`
- `max_bytes_to_copy`

buffer_copy (21 of 30 overloads)

Copies a limited number of bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const const_buffer & source,
    std::size_t max_bytes_to_copy);
```

Parameters

target	A modifiable buffer representing the memory region to which the bytes will be copied.
source	A non-modifiable buffer representing the memory region from which the bytes will be copied.
max_bytes_to_copy	The maximum number of bytes to be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`
- `max_bytes_to_copy`

buffer_copy (22 of 30 overloads)

Copies a limited number of bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const const_buffers_1 & source,
    std::size_t max_bytes_to_copy);
```

Parameters

target	A modifiable buffer representing the memory region to which the bytes will be copied.
source	A non-modifiable buffer representing the memory region from which the bytes will be copied.
max_bytes_to_copy	The maximum number of bytes to be copied.

Return Value

The number of bytes copied.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`
- `max_bytes_to_copy`

buffer_copy (23 of 30 overloads)

Copies a limited number of bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const mutable_buffer & source,
    std::size_t max_bytes_to_copy);
```

Parameters

target	A modifiable buffer representing the memory region to which the bytes will be copied.
source	A modifiable buffer representing the memory region from which the bytes will be copied. The contents of the source buffer will not be modified.
max_bytes_to_copy	The maximum number of bytes to be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`
- `max_bytes_to_copy`

buffer_copy (24 of 30 overloads)

Copies a limited number of bytes from a source buffer to a target buffer.

```
std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const mutable_buffers_1 & source,
    std::size_t max_bytes_to_copy);
```

Parameters

target	A modifiable buffer representing the memory region to which the bytes will be copied.
source	A modifiable buffer representing the memory region from which the bytes will be copied. The contents of the source buffer will not be modified.
max_bytes_to_copy	The maximum number of bytes to be copied.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`
- `max_bytes_to_copy`

buffer_copy (25 of 30 overloads)

Copies a limited number of bytes from a source buffer sequence to a target buffer.

```
template<
    typename ConstBufferSequence>
std::size_t buffer_copy(
    const mutable_buffers_1 & target,
    const ConstBufferSequence & source,
    std::size_t max_bytes_to_copy);
```

Parameters

target	A modifiable buffer representing the memory region to which the bytes will be copied.
source	A non-modifiable buffer sequence representing the memory regions from which the bytes will be copied.
max_bytes_to_copy	The maximum number of bytes to be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`
- `max_bytes_to_copy`

buffer_copy (26 of 30 overloads)

Copies a limited number of bytes from a source buffer to a target buffer sequence.

```
template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const const_buffer & source,
    std::size_t max_bytes_to_copy);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

target	A modifiable buffer sequence representing the memory regions to which the bytes will be copied.
source	A non-modifiable buffer representing the memory region from which the bytes will be copied.
max_bytes_to_copy	The maximum number of bytes to be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`
- `max_bytes_to_copy`

buffer_copy (27 of 30 overloads)

Copies a limited number of bytes from a source buffer to a target buffer sequence.

```
template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const const_buffers_1 & source,
    std::size_t max_bytes_to_copy);
```

Parameters

target	A modifiable buffer sequence representing the memory regions to which the bytes will be copied.
source	A non-modifiable buffer representing the memory region from which the bytes will be copied.
max_bytes_to_copy	The maximum number of bytes to be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`
- `max_bytes_to_copy`

buffer_copy (28 of 30 overloads)

Copies a limited number of bytes from a source buffer to a target buffer sequence.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const mutable_buffer & source,
    std::size_t max_bytes_to_copy);
```

Parameters

target	A modifiable buffer sequence representing the memory regions to which the bytes will be copied.
source	A modifiable buffer representing the memory region from which the bytes will be copied. The contents of the source buffer will not be modified.
max_bytes_to_copy	The maximum number of bytes to be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`
- `max_bytes_to_copy`

buffer_copy (29 of 30 overloads)

Copies a limited number of bytes from a source buffer to a target buffer sequence.

```
template<
    typename MutableBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const mutable_buffers_1 & source,
    std::size_t max_bytes_to_copy);
```

Parameters

target	A modifiable buffer sequence representing the memory regions to which the bytes will be copied.
source	A modifiable buffer representing the memory region from which the bytes will be copied. The contents of the source buffer will not be modified.
max_bytes_to_copy	The maximum number of bytes to be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- `buffer_size(source)`
- `max_bytes_to_copy`

buffer_copy (30 of 30 overloads)

Copies a limited number of bytes from a source buffer sequence to a target buffer sequence.

```
template<
    typename MutableBufferSequence,
    typename ConstBufferSequence>
std::size_t buffer_copy(
    const MutableBufferSequence & target,
    const ConstBufferSequence & source,
    std::size_t max_bytes_to_copy);
```

Parameters

target	A modifiable buffer sequence representing the memory regions to which the bytes will be copied.
source	A non-modifiable buffer sequence representing the memory regions from which the bytes will be copied.
max_bytes_to_copy	The maximum number of bytes to be copied.

Return Value

The number of bytes copied.

Remarks

The number of bytes copied is the lesser of:

- `buffer_size(target)`
- `buffer_size(source)`
- `max_bytes_to_copy`

buffer_size

The `boost::asio::buffer_size` function determines the total number of bytes in a buffer or buffer sequence.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t buffer_size(
    const mutable_buffer & b);
  » more...

std::size_t buffer_size(
    const mutable_buffers_1 & b);
  » more...

std::size_t buffer_size(
    const const_buffer & b);
  » more...

std::size_t buffer_size(
    const const_buffers_1 & b);
  » more...

template<
    typename BufferSequence>
std::size_t buffer_size(
    const BufferSequence & b);
  » more...
```

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

buffer_size (1 of 5 overloads)

Get the number of bytes in a modifiable buffer.

```
std::size_t buffer_size(
    const mutable_buffer & b);
```

buffer_size (2 of 5 overloads)

Get the number of bytes in a modifiable buffer.

```
std::size_t buffer_size(
    const mutable_buffers_1 & b);
```

buffer_size (3 of 5 overloads)

Get the number of bytes in a non-modifiable buffer.

```
std::size_t buffer_size(
    const const_buffer & b);
```

buffer_size (4 of 5 overloads)

Get the number of bytes in a non-modifiable buffer.

```
std::size_t buffer_size(
    const const_buffers_1 & b);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffer_size (5 of 5 overloads)

Get the total number of bytes in a buffer sequence.

```
template<
    typename BufferSequence>
std::size_t buffer_size(
    const BufferSequence & b);
```

The `BufferSequence` template parameter may meet either of the `ConstBufferSequence` or `MutableBufferSequence` type requirements.

buffered_read_stream

Adds buffering to the read-related operations of a stream.

```
template<
    typename Stream>
class buffered_read_stream :
  noncopyable
```

Types

Name	Description
lowest_layer_type	The type of the lowest layer.
next_layer_type	The type of the next layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
async_fill	Start an asynchronous fill.
async_read_some	Start an asynchronous read. The buffer into which the data will be read must be valid for the lifetime of the asynchronous operation.
async_write_some	Start an asynchronous write. The data being written must be valid for the lifetime of the asynchronous operation.
buffered_read_stream	Construct, passing the specified argument to initialise the next layer.
close	Close the stream.
fill	Fill the buffer with some data. Returns the number of bytes placed in the buffer as a result of the operation. Throws an exception on failure. Fill the buffer with some data. Returns the number of bytes placed in the buffer as a result of the operation, or 0 if an error occurred.
get_io_service	Get the io_service associated with the object.
in_avail	Determine the amount of data that may be read without blocking.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
next_layer	Get a reference to the next layer.
peek	Peek at the incoming data on the stream. Returns the number of bytes read. Throws an exception on failure. Peek at the incoming data on the stream. Returns the number of bytes read, or 0 if an error occurred.
read_some	Read some data from the stream. Returns the number of bytes read. Throws an exception on failure. Read some data from the stream. Returns the number of bytes read or 0 if an error occurred.
write_some	Write the given data to the stream. Returns the number of bytes written. Throws an exception on failure. Write the given data to the stream. Returns the number of bytes written, or 0 if an error occurred.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Data Members

Name	Description
default_buffer_size	The default buffer size.

The `buffered_read_stream` class template can be used to add buffering to the synchronous and asynchronous read operations of a stream.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/buffered_read_stream.hpp`

Convenience header: `boost/asio.hpp`

buffered_read_stream::async_fill

Start an asynchronous fill.

```
template<
    typename ReadHandler>
void async_fill(
    ReadHandler handler);
```

buffered_read_stream::async_read_some

Start an asynchronous read. The buffer into which the data will be read must be valid for the lifetime of the asynchronous operation.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_some(
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

buffered_read_stream::async_write_some

Start an asynchronous write. The data being written must be valid for the lifetime of the asynchronous operation.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_some(
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

buffered_read_stream::buffered_read_stream

Construct, passing the specified argument to initialise the next layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename Arg>
explicit buffered_read_stream(
    Arg & a);
  » more...

template<
    typename Arg>
buffered_read_stream(
    Arg & a,
    std::size_t buffer_size);
  » more...
```

buffered_read_stream::buffered_read_stream (1 of 2 overloads)

Construct, passing the specified argument to initialise the next layer.

```
template<
    typename Arg>
buffered_read_stream(
    Arg & a);
```

buffered_read_stream::buffered_read_stream (2 of 2 overloads)

Construct, passing the specified argument to initialise the next layer.

```
template<
    typename Arg>
buffered_read_stream(
    Arg & a,
    std::size_t buffer_size);
```

buffered_read_stream::close

Close the stream.

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

buffered_read_stream::close (1 of 2 overloads)

Close the stream.

```
void close();
```

buffered_read_stream::close (2 of 2 overloads)

Close the stream.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffered_read_stream::default_buffer_size

The default buffer size.

```
static const std::size_t default_buffer_size = implementation_defined;
```

buffered_read_stream::fill

Fill the buffer with some data. Returns the number of bytes placed in the buffer as a result of the operation. Throws an exception on failure.

```
std::size_t fill();
  » more...
```

Fill the buffer with some data. Returns the number of bytes placed in the buffer as a result of the operation, or 0 if an error occurred.

```
std::size_t fill(
    boost::system::error_code & ec);
  » more...
```

buffered_read_stream::fill (1 of 2 overloads)

Fill the buffer with some data. Returns the number of bytes placed in the buffer as a result of the operation. Throws an exception on failure.

```
std::size_t fill();
```

buffered_read_stream::fill (2 of 2 overloads)

Fill the buffer with some data. Returns the number of bytes placed in the buffer as a result of the operation, or 0 if an error occurred.

```
std::size_t fill(
    boost::system::error_code & ec);
```

buffered_read_stream::get_io_service

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

buffered_read_stream::in_avail

Determine the amount of data that may be read without blocking.

```
std::size_t in_avail();
  » more...

std::size_t in_avail(
    boost::system::error_code & ec);
  » more...
```

buffered_read_stream::in_avail (1 of 2 overloads)

Determine the amount of data that may be read without blocking.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t in_avail();
```

buffered_read_stream::in_avail (2 of 2 overloads)

Determine the amount of data that may be read without blocking.

```
std::size_t in_avail(
    boost::system::error_code & ec);
```

buffered_read_stream::lowest_layer

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

buffered_read_stream::lowest_layer (1 of 2 overloads)

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

buffered_read_stream::lowest_layer (2 of 2 overloads)

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

buffered_read_stream::lowest_layer_type

The type of the lowest layer.

```
typedef next_layer_type::lowest_layer_type lowest_layer_type;
```

Requirements

Header: `boost/asio/buffered_read_stream.hpp`

Convenience header: `boost/asio.hpp`

buffered_read_stream::next_layer

Get a reference to the next layer.

```
next_layer_type & next_layer();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffered_read_stream::next_layer_type

The type of the next layer.

```
typedef boost::remove_reference< Stream >::type next_layer_type;
```

Requirements

Header: `boost/asio/buffered_read_stream.hpp`

Convenience header: `boost/asio.hpp`

buffered_read_stream::peek

Peek at the incoming data on the stream. Returns the number of bytes read. Throws an exception on failure.

```
template<
    typename MutableBufferSequence>
std::size_t peek(
    const MutableBufferSequence & buffers);
  » more...
```

Peek at the incoming data on the stream. Returns the number of bytes read, or 0 if an error occurred.

```
template<
    typename MutableBufferSequence>
std::size_t peek(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

buffered_read_stream::peek (1 of 2 overloads)

Peek at the incoming data on the stream. Returns the number of bytes read. Throws an exception on failure.

```
template<
    typename MutableBufferSequence>
std::size_t peek(
    const MutableBufferSequence & buffers);
```

buffered_read_stream::peek (2 of 2 overloads)

Peek at the incoming data on the stream. Returns the number of bytes read, or 0 if an error occurred.

```
template<
    typename MutableBufferSequence>
std::size_t peek(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

buffered_read_stream::read_some

Read some data from the stream. Returns the number of bytes read. Throws an exception on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
  » more...
```

Read some data from the stream. Returns the number of bytes read or 0 if an error occurred.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

buffered_read_stream::read_some (1 of 2 overloads)

Read some data from the stream. Returns the number of bytes read. Throws an exception on failure.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
```

buffered_read_stream::read_some (2 of 2 overloads)

Read some data from the stream. Returns the number of bytes read or 0 if an error occurred.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

buffered_read_stream::write_some

Write the given data to the stream. Returns the number of bytes written. Throws an exception on failure.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
  » more...
```

Write the given data to the stream. Returns the number of bytes written, or 0 if an error occurred.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

buffered_read_stream::write_some (1 of 2 overloads)

Write the given data to the stream. Returns the number of bytes written. Throws an exception on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
```

buffered_read_stream::write_some (2 of 2 overloads)

Write the given data to the stream. Returns the number of bytes written, or 0 if an error occurred.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

buffered_stream

Adds buffering to the read- and write-related operations of a stream.

```
template<
    typename Stream>
class buffered_stream :
  noncopyable
```

Types

Name	Description
lowest_layer_type	The type of the lowest layer.
next_layer_type	The type of the next layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
async_fill	Start an asynchronous fill.
async_flush	Start an asynchronous flush.
async_read_some	Start an asynchronous read. The buffer into which the data will be read must be valid for the lifetime of the asynchronous operation.
async_write_some	Start an asynchronous write. The data being written must be valid for the lifetime of the asynchronous operation.
buffered_stream	Construct, passing the specified argument to initialise the next layer.
close	Close the stream.
fill	Fill the buffer with some data. Returns the number of bytes placed in the buffer as a result of the operation. Throws an exception on failure. Fill the buffer with some data. Returns the number of bytes placed in the buffer as a result of the operation, or 0 if an error occurred.
flush	Flush all data from the buffer to the next layer. Returns the number of bytes written to the next layer on the last write operation. Throws an exception on failure. Flush all data from the buffer to the next layer. Returns the number of bytes written to the next layer on the last write operation, or 0 if an error occurred.
get_io_service	Get the io_service associated with the object.
in_avail	Determine the amount of data that may be read without blocking.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
next_layer	Get a reference to the next layer.
peek	Peek at the incoming data on the stream. Returns the number of bytes read. Throws an exception on failure. Peek at the incoming data on the stream. Returns the number of bytes read, or 0 if an error occurred.
read_some	Read some data from the stream. Returns the number of bytes read. Throws an exception on failure. Read some data from the stream. Returns the number of bytes read or 0 if an error occurred.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
write_some	Write the given data to the stream. Returns the number of bytes written. Throws an exception on failure. Write the given data to the stream. Returns the number of bytes written, or 0 if an error occurred.

The `buffered_stream` class template can be used to add buffering to the synchronous and asynchronous read and write operations of a stream.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/buffered_stream.hpp`

Convenience header: `boost/asio.hpp`

buffered_stream::async_fill

Start an asynchronous fill.

```
template<
    typename ReadHandler>
void async_fill(
    ReadHandler handler);
```

buffered_stream::async_flush

Start an asynchronous flush.

```
template<
    typename WriteHandler>
void async_flush(
    WriteHandler handler);
```

buffered_stream::async_read_some

Start an asynchronous read. The buffer into which the data will be read must be valid for the lifetime of the asynchronous operation.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_some(
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

buffered_stream::async_write_some

Start an asynchronous write. The data being written must be valid for the lifetime of the asynchronous operation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_some(
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

buffered_stream::buffered_stream

Construct, passing the specified argument to initialise the next layer.

```
template<
    typename Arg>
explicit buffered_stream(
    Arg & a);
  » more...

template<
    typename Arg>
explicit buffered_stream(
    Arg & a,
    std::size_t read_buffer_size,
    std::size_t write_buffer_size);
  » more...
```

buffered_stream::buffered_stream (1 of 2 overloads)

Construct, passing the specified argument to initialise the next layer.

```
template<
    typename Arg>
buffered_stream(
    Arg & a);
```

buffered_stream::buffered_stream (2 of 2 overloads)

Construct, passing the specified argument to initialise the next layer.

```
template<
    typename Arg>
buffered_stream(
    Arg & a,
    std::size_t read_buffer_size,
    std::size_t write_buffer_size);
```

buffered_stream::close

Close the stream.

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffered_stream::close (1 of 2 overloads)

Close the stream.

```
void close();
```

buffered_stream::close (2 of 2 overloads)

Close the stream.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```

buffered_stream::fill

Fill the buffer with some data. Returns the number of bytes placed in the buffer as a result of the operation. Throws an exception on failure.

```
std::size_t fill();
  » more...
```

Fill the buffer with some data. Returns the number of bytes placed in the buffer as a result of the operation, or 0 if an error occurred.

```
std::size_t fill(
    boost::system::error_code & ec);
  » more...
```

buffered_stream::fill (1 of 2 overloads)

Fill the buffer with some data. Returns the number of bytes placed in the buffer as a result of the operation. Throws an exception on failure.

```
std::size_t fill();
```

buffered_stream::fill (2 of 2 overloads)

Fill the buffer with some data. Returns the number of bytes placed in the buffer as a result of the operation, or 0 if an error occurred.

```
std::size_t fill(
    boost::system::error_code & ec);
```

buffered_stream::flush

Flush all data from the buffer to the next layer. Returns the number of bytes written to the next layer on the last write operation. Throws an exception on failure.

```
std::size_t flush();
  » more...
```

Flush all data from the buffer to the next layer. Returns the number of bytes written to the next layer on the last write operation, or 0 if an error occurred.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t flush(
    boost::system::error_code & ec);
  » more...
```

buffered_stream::flush (1 of 2 overloads)

Flush all data from the buffer to the next layer. Returns the number of bytes written to the next layer on the last write operation. Throws an exception on failure.

```
std::size_t flush();
```

buffered_stream::flush (2 of 2 overloads)

Flush all data from the buffer to the next layer. Returns the number of bytes written to the next layer on the last write operation, or 0 if an error occurred.

```
std::size_t flush(
    boost::system::error_code & ec);
```

buffered_stream::get_io_service

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

buffered_stream::in_avail

Determine the amount of data that may be read without blocking.

```
std::size_t in_avail();
  » more...

std::size_t in_avail(
    boost::system::error_code & ec);
  » more...
```

buffered_stream::in_avail (1 of 2 overloads)

Determine the amount of data that may be read without blocking.

```
std::size_t in_avail();
```

buffered_stream::in_avail (2 of 2 overloads)

Determine the amount of data that may be read without blocking.

```
std::size_t in_avail(
    boost::system::error_code & ec);
```

buffered_stream::lowest_layer

Get a reference to the lowest layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

buffered_stream::lowest_layer (1 of 2 overloads)

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

buffered_stream::lowest_layer (2 of 2 overloads)

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

buffered_stream::lowest_layer_type

The type of the lowest layer.

```
typedef next_layer_type::lowest_layer_type lowest_layer_type;
```

Requirements

Header: `boost/asio/buffered_stream.hpp`

Convenience header: `boost/asio.hpp`

buffered_stream::next_layer

Get a reference to the next layer.

```
next_layer_type & next_layer();
```

buffered_stream::next_layer_type

The type of the next layer.

```
typedef boost::remove_reference< Stream >::type next_layer_type;
```

Requirements

Header: `boost/asio/buffered_stream.hpp`

Convenience header: `boost/asio.hpp`

buffered_stream::peek

Peek at the incoming data on the stream. Returns the number of bytes read. Throws an exception on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t peek(
    const MutableBufferSequence & buffers);
  » more...
```

Peek at the incoming data on the stream. Returns the number of bytes read, or 0 if an error occurred.

```
template<
    typename MutableBufferSequence>
std::size_t peek(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

buffered_stream::peek (1 of 2 overloads)

Peek at the incoming data on the stream. Returns the number of bytes read. Throws an exception on failure.

```
template<
    typename MutableBufferSequence>
std::size_t peek(
    const MutableBufferSequence & buffers);
```

buffered_stream::peek (2 of 2 overloads)

Peek at the incoming data on the stream. Returns the number of bytes read, or 0 if an error occurred.

```
template<
    typename MutableBufferSequence>
std::size_t peek(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

buffered_stream::read_some

Read some data from the stream. Returns the number of bytes read. Throws an exception on failure.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
  » more...
```

Read some data from the stream. Returns the number of bytes read or 0 if an error occurred.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

buffered_stream::read_some (1 of 2 overloads)

Read some data from the stream. Returns the number of bytes read. Throws an exception on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
```

buffered_stream::read_some (2 of 2 overloads)

Read some data from the stream. Returns the number of bytes read or 0 if an error occurred.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

buffered_stream::write_some

Write the given data to the stream. Returns the number of bytes written. Throws an exception on failure.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
  » more...
```

Write the given data to the stream. Returns the number of bytes written, or 0 if an error occurred.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

buffered_stream::write_some (1 of 2 overloads)

Write the given data to the stream. Returns the number of bytes written. Throws an exception on failure.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
```

buffered_stream::write_some (2 of 2 overloads)

Write the given data to the stream. Returns the number of bytes written, or 0 if an error occurred.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

buffered_write_stream

Adds buffering to the write-related operations of a stream.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename Stream>
class buffered_write_stream :
  noncopyable
```

Types

Name	Description
lowest_layer_type	The type of the lowest layer.
next_layer_type	The type of the next layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
async_flush	Start an asynchronous flush.
async_read_some	Start an asynchronous read. The buffer into which the data will be read must be valid for the lifetime of the asynchronous operation.
async_write_some	Start an asynchronous write. The data being written must be valid for the lifetime of the asynchronous operation.
buffered_write_stream	Construct, passing the specified argument to initialise the next layer.
close	Close the stream.
flush	Flush all data from the buffer to the next layer. Returns the number of bytes written to the next layer on the last write operation. Throws an exception on failure. Flush all data from the buffer to the next layer. Returns the number of bytes written to the next layer on the last write operation, or 0 if an error occurred.
get_io_service	Get the io_service associated with the object.
in_avail	Determine the amount of data that may be read without blocking.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
next_layer	Get a reference to the next layer.
peek	Peek at the incoming data on the stream. Returns the number of bytes read. Throws an exception on failure. Peek at the incoming data on the stream. Returns the number of bytes read, or 0 if an error occurred.
read_some	Read some data from the stream. Returns the number of bytes read. Throws an exception on failure. Read some data from the stream. Returns the number of bytes read or 0 if an error occurred.
write_some	Write the given data to the stream. Returns the number of bytes written. Throws an exception on failure. Write the given data to the stream. Returns the number of bytes written, or 0 if an error occurred and the error handler did not throw.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Data Members

Name	Description
default_buffer_size	The default buffer size.

The `buffered_write_stream` class template can be used to add buffering to the synchronous and asynchronous write operations of a stream.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/buffered_write_stream.hpp`

Convenience header: `boost/asio.hpp`

buffered_write_stream::async_flush

Start an asynchronous flush.

```
template<
    typename WriteHandler>
void async_flush(
    WriteHandler handler);
```

buffered_write_stream::async_read_some

Start an asynchronous read. The buffer into which the data will be read must be valid for the lifetime of the asynchronous operation.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_some(
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

buffered_write_stream::async_write_some

Start an asynchronous write. The data being written must be valid for the lifetime of the asynchronous operation.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_some(
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

buffered_write_stream::buffered_write_stream

Construct, passing the specified argument to initialise the next layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename Arg>
explicit buffered_write_stream(
    Arg & a);
  » more...

template<
    typename Arg>
buffered_write_stream(
    Arg & a,
    std::size_t buffer_size);
  » more...
```

buffered_write_stream::buffered_write_stream (1 of 2 overloads)

Construct, passing the specified argument to initialise the next layer.

```
template<
    typename Arg>
buffered_write_stream(
    Arg & a);
```

buffered_write_stream::buffered_write_stream (2 of 2 overloads)

Construct, passing the specified argument to initialise the next layer.

```
template<
    typename Arg>
buffered_write_stream(
    Arg & a,
    std::size_t buffer_size);
```

buffered_write_stream::close

Close the stream.

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

buffered_write_stream::close (1 of 2 overloads)

Close the stream.

```
void close();
```

buffered_write_stream::close (2 of 2 overloads)

Close the stream.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffered_write_stream::default_buffer_size

The default buffer size.

```
static const std::size_t default_buffer_size = implementation_defined;
```

buffered_write_stream::flush

Flush all data from the buffer to the next layer. Returns the number of bytes written to the next layer on the last write operation. Throws an exception on failure.

```
std::size_t flush();
  » more...
```

Flush all data from the buffer to the next layer. Returns the number of bytes written to the next layer on the last write operation, or 0 if an error occurred.

```
std::size_t flush(
    boost::system::error_code & ec);
  » more...
```

buffered_write_stream::flush (1 of 2 overloads)

Flush all data from the buffer to the next layer. Returns the number of bytes written to the next layer on the last write operation. Throws an exception on failure.

```
std::size_t flush();
```

buffered_write_stream::flush (2 of 2 overloads)

Flush all data from the buffer to the next layer. Returns the number of bytes written to the next layer on the last write operation, or 0 if an error occurred.

```
std::size_t flush(
    boost::system::error_code & ec);
```

buffered_write_stream::get_io_service

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

buffered_write_stream::in_avail

Determine the amount of data that may be read without blocking.

```
std::size_t in_avail();
  » more...

std::size_t in_avail(
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffered_write_stream::in_avail (1 of 2 overloads)

Determine the amount of data that may be read without blocking.

```
std::size_t in_avail();
```

buffered_write_stream::in_avail (2 of 2 overloads)

Determine the amount of data that may be read without blocking.

```
std::size_t in_avail(
    boost::system::error_code & ec);
```

buffered_write_stream::lowest_layer

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

buffered_write_stream::lowest_layer (1 of 2 overloads)

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

buffered_write_stream::lowest_layer (2 of 2 overloads)

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

buffered_write_stream::lowest_layer_type

The type of the lowest layer.

```
typedef next_layer_type::lowest_layer_type lowest_layer_type;
```

Requirements

Header: `boost/asio/buffered_write_stream.hpp`

Convenience header: `boost/asio.hpp`

buffered_write_stream::next_layer

Get a reference to the next layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
next_layer_type & next_layer();
```

buffered_write_stream::next_layer_type

The type of the next layer.

```
typedef boost::remove_reference< Stream >::type next_layer_type;
```

Requirements

Header: `boost/asio/buffered_write_stream.hpp`

Convenience header: `boost/asio.hpp`

buffered_write_stream::peek

Peek at the incoming data on the stream. Returns the number of bytes read. Throws an exception on failure.

```
template<
    typename MutableBufferSequence>
std::size_t peek(
    const MutableBufferSequence & buffers);
  » more...
```

Peek at the incoming data on the stream. Returns the number of bytes read, or 0 if an error occurred.

```
template<
    typename MutableBufferSequence>
std::size_t peek(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

buffered_write_stream::peek (1 of 2 overloads)

Peek at the incoming data on the stream. Returns the number of bytes read. Throws an exception on failure.

```
template<
    typename MutableBufferSequence>
std::size_t peek(
    const MutableBufferSequence & buffers);
```

buffered_write_stream::peek (2 of 2 overloads)

Peek at the incoming data on the stream. Returns the number of bytes read, or 0 if an error occurred.

```
template<
    typename MutableBufferSequence>
std::size_t peek(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

buffered_write_stream::read_some

Read some data from the stream. Returns the number of bytes read. Throws an exception on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
  » more...
```

Read some data from the stream. Returns the number of bytes read or 0 if an error occurred.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

buffered_write_stream::read_some (1 of 2 overloads)

Read some data from the stream. Returns the number of bytes read. Throws an exception on failure.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
```

buffered_write_stream::read_some (2 of 2 overloads)

Read some data from the stream. Returns the number of bytes read or 0 if an error occurred.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

buffered_write_stream::write_some

Write the given data to the stream. Returns the number of bytes written. Throws an exception on failure.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
  » more...
```

Write the given data to the stream. Returns the number of bytes written, or 0 if an error occurred and the error handler did not throw.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

buffered_write_stream::write_some (1 of 2 overloads)

Write the given data to the stream. Returns the number of bytes written. Throws an exception on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
```

buffered_write_stream::write_some (2 of 2 overloads)

Write the given data to the stream. Returns the number of bytes written, or 0 if an error occurred and the error handler did not throw.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

buffers_begin

Construct an iterator representing the beginning of the buffers' data.

```
template<
    typename BufferSequence>
buffers_iterator< BufferSequence > buffers_begin(
    const BufferSequence & buffers);
```

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_end

Construct an iterator representing the end of the buffers' data.

```
template<
    typename BufferSequence>
buffers_iterator< BufferSequence > buffers_end(
    const BufferSequence & buffers);
```

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator

A random access iterator over the bytes in a buffer sequence.

```
template<
    typename BufferSequence,
    typename ByteType = char>
class buffers_iterator
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
difference_type	The type used for the distance between two iterators.
iterator_category	The iterator category.
pointer	The type of the result of applying operator->() to the iterator.
reference	The type of the result of applying operator*() to the iterator.
value_type	The type of the value pointed to by the iterator.

Member Functions

Name	Description
begin	Construct an iterator representing the beginning of the buffers' data.
buffers_iterator	Default constructor. Creates an iterator in an undefined state.
end	Construct an iterator representing the end of the buffers' data.
operator *	Dereference an iterator.
operator++	Increment operator (prefix). Increment operator (postfix).
operator+=	Addition operator.
operator--	Decrement operator (prefix). Decrement operator (postfix).
operator-=	Subtraction operator.
operator->	Dereference an iterator.
operator[]	Access an individual element.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Friends

Name	Description
operator!=	Test two iterators for inequality.
operator+	Addition operator.
operator-	Subtraction operator.
operator<	Compare two iterators.
operator<=	Compare two iterators.
operator==	Test two iterators for equality.
operator>	Compare two iterators.
operator>=	Compare two iterators.

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::begin

Construct an iterator representing the beginning of the buffers' data.

```
static buffers_iterator begin(
    const BufferSequence & buffers);
```

buffers_iterator::buffers_iterator

Default constructor. Creates an iterator in an undefined state.

```
buffers_iterator();
```

buffers_iterator::difference_type

The type used for the distance between two iterators.

```
typedef std::ptrdiff_t difference_type;
```

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::end

Construct an iterator representing the end of the buffers' data.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
static buffers_iterator end(
    const BufferSequence & buffers);
```

buffers_iterator::iterator_category

The iterator category.

```
typedef std::random_access_iterator_tag iterator_category;
```

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::operator *

Dereference an iterator.

```
reference operator *() const;
```

buffers_iterator::operator!=

Test two iterators for inequality.

```
friend bool operator!=(
    const buffers_iterator & a,
    const buffers_iterator & b);
```

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::operator+

Addition operator.

```
friend buffers_iterator operator+(
    const buffers_iterator & iter,
    std::ptrdiff_t difference);
  » more...

friend buffers_iterator operator+(
    std::ptrdiff_t difference,
    const buffers_iterator & iter);
  » more...
```

buffers_iterator::operator+ (1 of 2 overloads)

Addition operator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
friend buffers_iterator operator+(
    const buffers_iterator & iter,
    std::ptrdiff_t difference);
```

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::operator+ (2 of 2 overloads)

Addition operator.

```
friend buffers_iterator operator+(
    std::ptrdiff_t difference,
    const buffers_iterator & iter);
```

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::operator++

Increment operator (prefix).

```
buffers_iterator & operator++();
  » more...
```

Increment operator (postfix).

```
buffers_iterator operator++(
    int );
  » more...
```

buffers_iterator::operator++ (1 of 2 overloads)

Increment operator (prefix).

```
buffers_iterator & operator++();
```

buffers_iterator::operator++ (2 of 2 overloads)

Increment operator (postfix).

```
buffers_iterator operator++(
    int );
```

buffers_iterator::operator+=

Addition operator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
buffers_iterator & operator+=(
    std::ptrdiff_t difference);
```

buffers_iterator::operator-

Subtraction operator.

```
friend buffers_iterator operator-(
    const buffers_iterator & iter,
    std::ptrdiff_t difference);
  » more...

friend std::ptrdiff_t operator-(
    const buffers_iterator & a,
    const buffers_iterator & b);
  » more...
```

buffers_iterator::operator- (1 of 2 overloads)

Subtraction operator.

```
friend buffers_iterator operator-(
    const buffers_iterator & iter,
    std::ptrdiff_t difference);
```

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::operator- (2 of 2 overloads)

Subtraction operator.

```
friend std::ptrdiff_t operator-(
    const buffers_iterator & a,
    const buffers_iterator & b);
```

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::operator--

Decrement operator (prefix).

```
buffers_iterator & operator--();
  » more...
```

Decrement operator (postfix).

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
buffers_iterator operator--(
    int );
  » more...
```

buffers_iterator::operator-- (1 of 2 overloads)

Decrement operator (prefix).

```
buffers_iterator & operator--();
```

buffers_iterator::operator-- (2 of 2 overloads)

Decrement operator (postfix).

```
buffers_iterator operator--(
    int );
```

buffers_iterator::operator-=

Subtraction operator.

```
buffers_iterator & operator-=(
    std::ptrdiff_t difference);
```

buffers_iterator::operator->

Dereference an iterator.

```
pointer operator->() const;
```

buffers_iterator::operator<

Compare two iterators.

```
friend bool operator<(
    const buffers_iterator & a,
    const buffers_iterator & b);
```

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::operator<=

Compare two iterators.

```
friend bool operator<=(
    const buffers_iterator & a,
    const buffers_iterator & b);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::operator==

Test two iterators for equality.

```
friend bool operator==(
    const buffers_iterator & a,
    const buffers_iterator & b);
```

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::operator>

Compare two iterators.

```
friend bool operator>(
    const buffers_iterator & a,
    const buffers_iterator & b);
```

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::operator>=

Compare two iterators.

```
friend bool operator>=(
    const buffers_iterator & a,
    const buffers_iterator & b);
```

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::operator[]

Access an individual element.

```
reference operator[](
    std::ptrdiff_t difference) const;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffers_iterator::pointer

The type of the result of applying `operator->()` to the iterator.

```
typedef const_or_non_const_ByteType * pointer;
```

If the buffer sequence stores buffer objects that are convertible to `mutable_buffer`, this is a pointer to a non-const ByteType. Otherwise, a pointer to a const ByteType.

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::reference

The type of the result of applying `operator*()` to the iterator.

```
typedef const_or_non_const_ByteType & reference;
```

If the buffer sequence stores buffer objects that are convertible to `mutable_buffer`, this is a reference to a non-const ByteType. Otherwise, a reference to a const ByteType.

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

buffers_iterator::value_type

The type of the value pointed to by the iterator.

```
typedef ByteType value_type;
```

Requirements

Header: `boost/asio/buffers_iterator.hpp`

Convenience header: `boost/asio.hpp`

connect

Establishes a socket connection by trying each endpoint in a sequence.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin);
  » more...

template<
    typename Protocol,
    typename SocketService,
    typename Iterator>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    boost::system::error_code & ec);
  » more...

template<
    typename Protocol,
    typename SocketService,
    typename Iterator>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    Iterator end);
  » more...

template<
    typename Protocol,
    typename SocketService,
    typename Iterator>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    Iterator end,
    boost::system::error_code & ec);
  » more...

template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ConnectCondition>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    ConnectCondition connect_condition);
  » more...

template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ConnectCondition>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    ConnectCondition connect_condition,
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ConnectCondition>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    Iterator end,
    ConnectCondition connect_condition);
  » more...

template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ConnectCondition>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    Iterator end,
    ConnectCondition connect_condition,
    boost::system::error_code & ec);
  » more...
```

Requirements

Header: `boost/asio/connect.hpp`

Convenience header: `boost/asio.hpp`

connect (1 of 8 overloads)

Establishes a socket connection by trying each endpoint in a sequence.

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin);
```

This function attempts to connect a socket to one of a sequence of endpoints. It does this by repeated calls to the socket's `connect` member function, once for each endpoint in the sequence, until a connection is successfully established.

Parameters

s — The socket to be connected. If the socket is already open, it will be closed.

begin — An iterator pointing to the start of a sequence of endpoints.

Return Value

On success, an iterator denoting the successfully connected endpoint. Otherwise, the end iterator.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure. If the sequence is empty, the associated `error_code` is `boost::asio::error::not_found`. Otherwise, contains the error from the last connection attempt.

Remarks

This overload assumes that a default constructed object of type `Iterator` represents the end of the sequence. This is a valid assumption for iterator types such as `boost::asio::ip::tcp::resolver::iterator`.

Example

```
tcp::resolver r(io_service);
tcp::resolver::query q("host", "service");
tcp::socket s(io_service);
boost::asio::connect(s, r.resolve(q));
```

connect (2 of 8 overloads)

Establishes a socket connection by trying each endpoint in a sequence.

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    boost::system::error_code & ec);
```

This function attempts to connect a socket to one of a sequence of endpoints. It does this by repeated calls to the socket's `connect` member function, once for each endpoint in the sequence, until a connection is successfully established.

Parameters

s	The socket to be connected. If the socket is already open, it will be closed.
begin	An iterator pointing to the start of a sequence of endpoints.
ec	Set to indicate what error occurred, if any. If the sequence is empty, set to `boost::asio::error::not_found`. Otherwise, contains the error from the last connection attempt.

Return Value

On success, an iterator denoting the successfully connected endpoint. Otherwise, the end iterator.

Remarks

This overload assumes that a default constructed object of type `Iterator` represents the end of the sequence. This is a valid assumption for iterator types such as `boost::asio::ip::tcp::resolver::iterator`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

```
tcp::resolver r(io_service);
tcp::resolver::query q("host", "service");
tcp::socket s(io_service);
boost::system::error_code ec;
boost::asio::connect(s, r.resolve(q), ec);
if (ec)
{
  // An error occurred.
}
```

connect (3 of 8 overloads)

Establishes a socket connection by trying each endpoint in a sequence.

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    Iterator end);
```

This function attempts to connect a socket to one of a sequence of endpoints. It does this by repeated calls to the socket's `connect` member function, once for each endpoint in the sequence, until a connection is successfully established.

Parameters

s	The socket to be connected. If the socket is already open, it will be closed.
begin	An iterator pointing to the start of a sequence of endpoints.
end	An iterator pointing to the end of a sequence of endpoints.

Return Value

On success, an iterator denoting the successfully connected endpoint. Otherwise, the end iterator.

Exceptions

boost::system::system_error	Thrown on failure. If the sequence is empty, the associated `error_code` is `boost::asio::error::not_found`. Otherwise, contains the error from the last connection attempt.

Example

```
tcp::resolver r(io_service);
tcp::resolver::query q("host", "service");
tcp::resolver::iterator i = r.resolve(q), end;
tcp::socket s(io_service);
boost::asio::connect(s, i, end);
```

connect (4 of 8 overloads)

Establishes a socket connection by trying each endpoint in a sequence.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    Iterator end,
    boost::system::error_code & ec);
```

This function attempts to connect a socket to one of a sequence of endpoints. It does this by repeated calls to the socket's `connect` member function, once for each endpoint in the sequence, until a connection is successfully established.

Parameters

s	The socket to be connected. If the socket is already open, it will be closed.
begin	An iterator pointing to the start of a sequence of endpoints.
end	An iterator pointing to the end of a sequence of endpoints.
ec	Set to indicate what error occurred, if any. If the sequence is empty, set to `boost::asio::error::not_found`. Otherwise, contains the error from the last connection attempt.

Return Value

On success, an iterator denoting the successfully connected endpoint. Otherwise, the end iterator.

Example

```
tcp::resolver r(io_service);
tcp::resolver::query q("host", "service");
tcp::resolver::iterator i = r.resolve(q), end;
tcp::socket s(io_service);
boost::system::error_code ec;
boost::asio::connect(s, i, end, ec);
if (ec)
{
  // An error occurred.
}
```

connect (5 of 8 overloads)

Establishes a socket connection by trying each endpoint in a sequence.

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ConnectCondition>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    ConnectCondition connect_condition);
```

This function attempts to connect a socket to one of a sequence of endpoints. It does this by repeated calls to the socket's `connect` member function, once for each endpoint in the sequence, until a connection is successfully established.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

s	The socket to be connected. If the socket is already open, it will be closed.
begin	An iterator pointing to the start of a sequence of endpoints.
connect_condition	A function object that is called prior to each connection attempt. The signature of the function object must be:

```
Iterator connect_condition(
    const boost::system::error_code& ec,
    Iterator next);
```

The `ec` parameter contains the result from the most recent connect operation. Before the first connection attempt, `ec` is always set to indicate success. The `next` parameter is an iterator pointing to the next endpoint to be tried. The function object should return the next iterator, but is permitted to return a different iterator so that endpoints may be skipped. The implementation guarantees that the function object will never be called with the end iterator.

Return Value

On success, an iterator denoting the successfully connected endpoint. Otherwise, the end iterator.

Exceptions

boost::system::system_error	Thrown on failure. If the sequence is empty, the associated `error_code` is `boost::asio::error::not_found`. Otherwise, contains the error from the last connection attempt.

Remarks

This overload assumes that a default constructed object of type `Iterator` represents the end of the sequence. This is a valid assumption for iterator types such as `boost::asio::ip::tcp::resolver::iterator`.

Example

The following connect condition function object can be used to output information about the individual connection attempts:

```
struct my_connect_condition
{
  template <typename Iterator>
  Iterator operator()(
      const boost::system::error_code& ec,
      Iterator next)
  {
    if (ec) std::cout << "Error: " << ec.message() << std::endl;
    std::cout << "Trying: " << next->endpoint() << std::endl;
    return next;
  }
};
```

It would be used with the `boost::asio::connect` function as follows:

```
tcp::resolver r(io_service);
tcp::resolver::query q("host", "service");
tcp::socket s(io_service);
tcp::resolver::iterator i = boost::asio::connect(
    s, r.resolve(q), my_connect_condition());
std::cout << "Connected to: " << i->endpoint() << std::endl;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

connect (6 of 8 overloads)

Establishes a socket connection by trying each endpoint in a sequence.

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ConnectCondition>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    ConnectCondition connect_condition,
    boost::system::error_code & ec);
```

This function attempts to connect a socket to one of a sequence of endpoints. It does this by repeated calls to the socket's `connect` member function, once for each endpoint in the sequence, until a connection is successfully established.

Parameters

s — The socket to be connected. If the socket is already open, it will be closed.

begin — An iterator pointing to the start of a sequence of endpoints.

connect_condition — A function object that is called prior to each connection attempt. The signature of the function object must be:

```
Iterator connect_condition(
    const boost::system::error_code& ec,
    Iterator next);
```

The `ec` parameter contains the result from the most recent connect operation. Before the first connection attempt, `ec` is always set to indicate success. The `next` parameter is an iterator pointing to the next endpoint to be tried. The function object should return the next iterator, but is permitted to return a different iterator so that endpoints may be skipped. The implementation guarantees that the function object will never be called with the end iterator.

ec — Set to indicate what error occurred, if any. If the sequence is empty, set to `boost::asio::error::not_found`. Otherwise, contains the error from the last connection attempt.

Return Value

On success, an iterator denoting the successfully connected endpoint. Otherwise, the end iterator.

Remarks

This overload assumes that a default constructed object of type `Iterator` represents the end of the sequence. This is a valid assumption for iterator types such as `boost::asio::ip::tcp::resolver::iterator`.

Example

The following connect condition function object can be used to output information about the individual connection attempts:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
struct my_connect_condition
{
  template <typename Iterator>
  Iterator operator()(
      const boost::system::error_code& ec,
      Iterator next)
  {
    if (ec) std::cout << "Error: " << ec.message() << std::endl;
    std::cout << "Trying: " << next->endpoint() << std::endl;
    return next;
  }
};
```

It would be used with the `boost::asio::connect` function as follows:

```
tcp::resolver r(io_service);
tcp::resolver::query q("host", "service");
tcp::socket s(io_service);
boost::system::error_code ec;
tcp::resolver::iterator i = boost::asio::connect(
    s, r.resolve(q), my_connect_condition(), ec);
if (ec)
{
  // An error occurred.
}
else
{
  std::cout << "Connected to: " << i->endpoint() << std::endl;
}
```

connect (7 of 8 overloads)

Establishes a socket connection by trying each endpoint in a sequence.

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ConnectCondition>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    Iterator end,
    ConnectCondition connect_condition);
```

This function attempts to connect a socket to one of a sequence of endpoints. It does this by repeated calls to the socket's `connect` member function, once for each endpoint in the sequence, until a connection is successfully established.

Parameters

s	The socket to be connected. If the socket is already open, it will be closed.
begin	An iterator pointing to the start of a sequence of endpoints.
end	An iterator pointing to the end of a sequence of endpoints.
connect_condition	A function object that is called prior to each connection attempt. The signature of the function object must be:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
Iterator connect_condition(
    const boost::system::error_code& ec,
    Iterator next);
```

The `ec` parameter contains the result from the most recent connect operation. Before the first connection attempt, `ec` is always set to indicate success. The `next` parameter is an iterator pointing to the next endpoint to be tried. The function object should return the next iterator, but is permitted to return a different iterator so that endpoints may be skipped. The implementation guarantees that the function object will never be called with the end iterator.

Return Value

On success, an iterator denoting the successfully connected endpoint. Otherwise, the end iterator.

Exceptions

boost::system::system_error	Thrown on failure. If the sequence is empty, the associated `error_code` is `boost::asio::error::not_found`. Otherwise, contains the error from the last connection attempt.

Example

The following connect condition function object can be used to output information about the individual connection attempts:

```
struct my_connect_condition
{
  template <typename Iterator>
  Iterator operator()(
      const boost::system::error_code& ec,
      Iterator next)
  {
    if (ec) std::cout << "Error: " << ec.message() << std::endl;
    std::cout << "Trying: " << next->endpoint() << std::endl;
    return next;
  }
};
```

It would be used with the `boost::asio::connect` function as follows:

```
tcp::resolver r(io_service);
tcp::resolver::query q("host", "service");
tcp::resolver::iterator i = r.resolve(q), end;
tcp::socket s(io_service);
i = boost::asio::connect(s, i, end, my_connect_condition());
std::cout << "Connected to: " << i->endpoint() << std::endl;
```

connect (8 of 8 overloads)

Establishes a socket connection by trying each endpoint in a sequence.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename Protocol,
    typename SocketService,
    typename Iterator,
    typename ConnectCondition>
Iterator connect(
    basic_socket< Protocol, SocketService > & s,
    Iterator begin,
    Iterator end,
    ConnectCondition connect_condition,
    boost::system::error_code & ec);
```

This function attempts to connect a socket to one of a sequence of endpoints. It does this by repeated calls to the socket's `connect` member function, once for each endpoint in the sequence, until a connection is successfully established.

Parameters

s — The socket to be connected. If the socket is already open, it will be closed.

begin — An iterator pointing to the start of a sequence of endpoints.

end — An iterator pointing to the end of a sequence of endpoints.

connect_condition — A function object that is called prior to each connection attempt. The signature of the function object must be:

```
Iterator connect_condition(
    const boost::system::error_code& ec,
    Iterator next);
```

The `ec` parameter contains the result from the most recent connect operation. Before the first connection attempt, `ec` is always set to indicate success. The `next` parameter is an iterator pointing to the next endpoint to be tried. The function object should return the next iterator, but is permitted to return a different iterator so that endpoints may be skipped. The implementation guarantees that the function object will never be called with the end iterator.

ec — Set to indicate what error occurred, if any. If the sequence is empty, set to `boost::asio::error::not_found`. Otherwise, contains the error from the last connection attempt.

Return Value

On success, an iterator denoting the successfully connected endpoint. Otherwise, the end iterator.

Example

The following connect condition function object can be used to output information about the individual connection attempts:

```
struct my_connect_condition
{
  template <typename Iterator>
  Iterator operator()(
      const boost::system::error_code& ec,
      Iterator next)
  {
    if (ec) std::cout << "Error: " << ec.message() << std::endl;
    std::cout << "Trying: " << next->endpoint() << std::endl;
    return next;
  }
};
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

It would be used with the `boost::asio::connect` function as follows:

```
tcp::resolver r(io_service);
tcp::resolver::query q("host", "service");
tcp::resolver::iterator i = r.resolve(q), end;
tcp::socket s(io_service);
boost::system::error_code ec;
i = boost::asio::connect(s, i, end, my_connect_condition(), ec);
if (ec)
{
  // An error occurred.
}
else
{
  std::cout << "Connected to: " << i->endpoint() << std::endl;
}
```

const_buffer

Holds a buffer that cannot be modified.

```
class const_buffer
```

Member Functions

Name	Description
const_buffer	Construct an empty buffer. Construct a buffer to represent a given memory range. Construct a non-modifiable buffer from a modifiable one.

Related Functions

Name	Description
operator+	Create a new non-modifiable buffer that is offset from the start of another.

The `const_buffer` class provides a safe representation of a buffer that cannot be modified. It does not own the underlying data, and so is cheap to copy or assign.

Accessing Buffer Contents

The contents of a buffer may be accessed using the `buffer_size` and `buffer_cast` functions:

```
boost::asio::const_buffer b1 = ...;
std::size_t s1 = boost::asio::buffer_size(b1);
const unsigned char* p1 = boost::asio::buffer_cast<const unsigned char*>(b1);
```

The `boost::asio::buffer_cast` function permits violations of type safety, so uses of it in application code should be carefully considered.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

const_buffer::const_buffer

Construct an empty buffer.

```
const_buffer();
  » more...
```

Construct a buffer to represent a given memory range.

```
const_buffer(
    const void * data,
    std::size_t size);
  » more...
```

Construct a non-modifiable buffer from a modifiable one.

```
const_buffer(
    const mutable_buffer & b);
  » more...
```

const_buffer::const_buffer (1 of 3 overloads)

Construct an empty buffer.

```
const_buffer();
```

const_buffer::const_buffer (2 of 3 overloads)

Construct a buffer to represent a given memory range.

```
const_buffer(
    const void * data,
    std::size_t size);
```

const_buffer::const_buffer (3 of 3 overloads)

Construct a non-modifiable buffer from a modifiable one.

```
const_buffer(
    const mutable_buffer & b);
```

const_buffer::operator+

Create a new non-modifiable buffer that is offset from the start of another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
const_buffer operator+(
    const const_buffer & b,
    std::size_t start);
  » more...

const_buffer operator+(
    std::size_t start,
    const const_buffer & b);
  » more...
```

const_buffer::operator+ (1 of 2 overloads)

Create a new non-modifiable buffer that is offset from the start of another.

```
const_buffer operator+(
    const const_buffer & b,
    std::size_t start);
```

const_buffer::operator+ (2 of 2 overloads)

Create a new non-modifiable buffer that is offset from the start of another.

```
const_buffer operator+(
    std::size_t start,
    const const_buffer & b);
```

const_buffers_1

Adapts a single non-modifiable buffer so that it meets the requirements of the ConstBufferSequence concept.

```
class const_buffers_1 :
  public const_buffer
```

Types

Name	Description
const_iterator	A random-access iterator type that may be used to read elements.
value_type	The type for each element in the list of buffers.

Member Functions

Name	Description
begin	Get a random-access iterator to the first element.
const_buffers_1	Construct to represent a given memory range. Construct to represent a single non-modifiable buffer.
end	Get a random-access iterator for one past the last element.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Related Functions

Name	Description
operator+	Create a new non-modifiable buffer that is offset from the start of another.

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

const_buffers_1::begin

Get a random-access iterator to the first element.

```
const_iterator begin() const;
```

const_buffers_1::const_buffers_1

Construct to represent a given memory range.

```
const_buffers_1(
    const void * data,
    std::size_t size);
  » more...
```

Construct to represent a single non-modifiable buffer.

```
explicit const_buffers_1(
    const const_buffer & b);
  » more...
```

const_buffers_1::const_buffers_1 (1 of 2 overloads)

Construct to represent a given memory range.

```
const_buffers_1(
    const void * data,
    std::size_t size);
```

const_buffers_1::const_buffers_1 (2 of 2 overloads)

Construct to represent a single non-modifiable buffer.

```
const_buffers_1(
    const const_buffer & b);
```

const_buffers_1::const_iterator

A random-access iterator type that may be used to read elements.

```
typedef const const_buffer * const_iterator;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

const_buffers_1::end

Get a random-access iterator for one past the last element.

```
const_iterator end() const;
```

const_buffers_1::operator+

Create a new non-modifiable buffer that is offset from the start of another.

```
const_buffer operator+(
    const const_buffer & b,
    std::size_t start);
  » more...

const_buffer operator+(
    std::size_t start,
    const const_buffer & b);
  » more...
```

const_buffers_1::operator+ (1 of 2 overloads)

Inherited from const_buffer.

Create a new non-modifiable buffer that is offset from the start of another.

```
const_buffer operator+(
    const const_buffer & b,
    std::size_t start);
```

const_buffers_1::operator+ (2 of 2 overloads)

Inherited from const_buffer.

Create a new non-modifiable buffer that is offset from the start of another.

```
const_buffer operator+(
    std::size_t start,
    const const_buffer & b);
```

const_buffers_1::value_type

The type for each element in the list of buffers.

```
typedef const_buffer value_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
const_buffer	Construct an empty buffer. Construct a buffer to represent a given memory range. Construct a non-modifiable buffer from a modifiable one.

Related Functions

Name	Description
operator+	Create a new non-modifiable buffer that is offset from the start of another.

The `const_buffer` class provides a safe representation of a buffer that cannot be modified. It does not own the underlying data, and so is cheap to copy or assign.

Accessing Buffer Contents

The contents of a buffer may be accessed using the `buffer_size` and `buffer_cast` functions:

```
boost::asio::const_buffer b1 = ...;
std::size_t s1 = boost::asio::buffer_size(b1);
const unsigned char* p1 = boost::asio::buffer_cast<const unsigned char*>(b1);
```

The `boost::asio::buffer_cast` function permits violations of type safety, so uses of it in application code should be carefully considered.

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

datagram_socket_service

Default service implementation for a datagram socket.

```
template<
    typename Protocol>
class datagram_socket_service :
  public io_service::service
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
endpoint_type	The endpoint type.
implementation_type	The type of a datagram socket.
native_handle_type	The native socket type.
native_type	(Deprecated: Use native_handle_type.) The native socket type.
protocol_type	The protocol type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to a datagram socket.
async_connect	Start an asynchronous connect.
async_receive	Start an asynchronous receive.
async_receive_from	Start an asynchronous receive that will get the endpoint of the sender.
async_send	Start an asynchronous send.
async_send_to	Start an asynchronous send.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
bind	
cancel	Cancel all asynchronous operations associated with the socket.
close	Close a datagram socket implementation.
connect	Connect the datagram socket to the specified endpoint.
construct	Construct a new datagram socket implementation.
datagram_socket_service	Construct a new datagram socket service for the specified io_service.
destroy	Destroy a datagram socket implementation.
get_io_service	Get the io_service object that owns the service.
get_option	Get a socket option.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint.
move_assign	Move-assign from another datagram socket implementation.
move_construct	Move-construct a new datagram socket implementation.
native	(Deprecated: Use native_handle().) Get the native socket implementation.
native_handle	Get the native socket implementation.
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	
receive	Receive some data from the peer.
receive_from	Receive a datagram with the endpoint of the sender.
remote_endpoint	Get the remote endpoint.
send	Send the given data to the peer.
send_to	Send a datagram to the specified endpoint.
set_option	Set a socket option.
shutdown	Disable sends or receives on the socket.

Data Members

Name	Description
id	The unique service identifier.

Requirements

Header: `boost/asio/datagram_socket_service.hpp`

Convenience header: `boost/asio.hpp`

datagram_socket_service::assign

Assign an existing native socket to a datagram socket.

```
boost::system::error_code assign(
    implementation_type & impl,
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
```

datagram_socket_service::async_connect

Start an asynchronous connect.

```
template<
    typename ConnectHandler>
void async_connect(
    implementation_type & impl,
    const endpoint_type & peer_endpoint,
    ConnectHandler handler);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

datagram_socket_service::async_receive

Start an asynchronous receive.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    ReadHandler handler);
```

datagram_socket_service::async_receive_from

Start an asynchronous receive that will get the endpoint of the sender.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive_from(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags,
    ReadHandler handler);
```

datagram_socket_service::async_send

Start an asynchronous send.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    WriteHandler handler);
```

datagram_socket_service::async_send_to

Start an asynchronous send.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send_to(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags,
    WriteHandler handler);
```

datagram_socket_service::at_mark

Determine whether the socket is at the out-of-band data mark.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
bool at_mark(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

datagram_socket_service::available

Determine the number of bytes available for reading.

```
std::size_t available(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

datagram_socket_service::bind

```
boost::system::error_code bind(
    implementation_type & impl,
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
```

datagram_socket_service::cancel

Cancel all asynchronous operations associated with the socket.

```
boost::system::error_code cancel(
    implementation_type & impl,
    boost::system::error_code & ec);
```

datagram_socket_service::close

Close a datagram socket implementation.

```
boost::system::error_code close(
    implementation_type & impl,
    boost::system::error_code & ec);
```

datagram_socket_service::connect

Connect the datagram socket to the specified endpoint.

```
boost::system::error_code connect(
    implementation_type & impl,
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
```

datagram_socket_service::construct

Construct a new datagram socket implementation.

```
void construct(
    implementation_type & impl);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

datagram_socket_service::datagram_socket_service

Construct a new datagram socket service for the specified `io_service`.

```
datagram_socket_service(
    boost::asio::io_service & io_service);
```

datagram_socket_service::destroy

Destroy a datagram socket implementation.

```
void destroy(
    implementation_type & impl);
```

datagram_socket_service::endpoint_type

The endpoint type.

```
typedef Protocol::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/datagram_socket_service.hpp`

Convenience header: `boost/asio.hpp`

datagram_socket_service::get_io_service

Inherited from io_service.

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

datagram_socket_service::get_option

Get a socket option.

```
template<
    typename GettableSocketOption>
boost::system::error_code get_option(
    const implementation_type & impl,
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
```

datagram_socket_service::id

The unique service identifier.

```
static boost::asio::io_service::id id;
```

datagram_socket_service::implementation_type

The type of a datagram socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef implementation_defined implementation_type;
```

Requirements

Header: `boost/asio/datagram_socket_service.hpp`

Convenience header: `boost/asio.hpp`

datagram_socket_service::io_control

Perform an IO control command on the socket.

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    implementation_type & impl,
    IoControlCommand & command,
    boost::system::error_code & ec);
```

datagram_socket_service::is_open

Determine whether the socket is open.

```
bool is_open(
    const implementation_type & impl) const;
```

datagram_socket_service::local_endpoint

Get the local endpoint.

```
endpoint_type local_endpoint(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

datagram_socket_service::move_assign

Move-assign from another datagram socket implementation.

```
void move_assign(
    implementation_type & impl,
    datagram_socket_service & other_service,
    implementation_type & other_impl);
```

datagram_socket_service::move_construct

Move-construct a new datagram socket implementation.

```
void move_construct(
    implementation_type & impl,
    implementation_type & other_impl);
```

datagram_socket_service::native

(Deprecated: Use `native_handle()`.) Get the native socket implementation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
native_type native(
    implementation_type & impl);
```

datagram_socket_service::native_handle

Get the native socket implementation.

```
native_handle_type native_handle(
    implementation_type & impl);
```

datagram_socket_service::native_handle_type

The native socket type.

```
typedef implementation_defined native_handle_type;
```

Requirements

Header: `boost/asio/datagram_socket_service.hpp`

Convenience header: `boost/asio.hpp`

datagram_socket_service::native_non_blocking

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking(
    const implementation_type & impl) const;
  » more...
```

Sets the non-blocking mode of the native socket implementation.

```
boost::system::error_code native_non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
  » more...
```

datagram_socket_service::native_non_blocking (1 of 2 overloads)

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking(
    const implementation_type & impl) const;
```

datagram_socket_service::native_non_blocking (2 of 2 overloads)

Sets the non-blocking mode of the native socket implementation.

```
boost::system::error_code native_non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

datagram_socket_service::native_type

(Deprecated: Use native_handle_type.) The native socket type.

```
typedef implementation_defined native_type;
```

Requirements

Header: `boost/asio/datagram_socket_service.hpp`

Convenience header: `boost/asio.hpp`

datagram_socket_service::non_blocking

Gets the non-blocking mode of the socket.

```
bool non_blocking(
    const implementation_type & impl) const;
  » more...
```

Sets the non-blocking mode of the socket.

```
boost::system::error_code non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
  » more...
```

datagram_socket_service::non_blocking (1 of 2 overloads)

Gets the non-blocking mode of the socket.

```
bool non_blocking(
    const implementation_type & impl) const;
```

datagram_socket_service::non_blocking (2 of 2 overloads)

Sets the non-blocking mode of the socket.

```
boost::system::error_code non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
```

datagram_socket_service::open

```
boost::system::error_code open(
    implementation_type & impl,
    const protocol_type & protocol,
    boost::system::error_code & ec);
```

datagram_socket_service::protocol_type

The protocol type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef Protocol protocol_type;
```

Requirements

Header: `boost/asio/datagram_socket_service.hpp`

Convenience header: `boost/asio.hpp`

datagram_socket_service::receive

Receive some data from the peer.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

datagram_socket_service::receive_from

Receive a datagram with the endpoint of the sender.

```
template<
    typename MutableBufferSequence>
std::size_t receive_from(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

datagram_socket_service::remote_endpoint

Get the remote endpoint.

```
endpoint_type remote_endpoint(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

datagram_socket_service::send

Send the given data to the peer.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

datagram_socket_service::send_to

Send a datagram to the specified endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t send_to(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

datagram_socket_service::set_option

Set a socket option.

```
template<
    typename SettableSocketOption>
boost::system::error_code set_option(
    implementation_type & impl,
    const SettableSocketOption & option,
    boost::system::error_code & ec);
```

datagram_socket_service::shutdown

Disable sends or receives on the socket.

```
boost::system::error_code shutdown(
    implementation_type & impl,
    socket_base::shutdown_type what,
    boost::system::error_code & ec);
```

deadline_timer

Typedef for the typical usage of timer. Uses a UTC clock.

```
typedef basic_deadline_timer< boost::posix_time::ptime > deadline_timer;
```

Types

Name	Description
duration_type	The duration type.
implementation_type	The underlying implementation type of I/O object.
service_type	The type of the service that will be used to provide I/O operations.
time_type	The time type.
traits_type	The time traits type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
async_wait	Start an asynchronous wait on the timer.
basic_deadline_timer	Constructor. Constructor to set a particular expiry time as an absolute time. Constructor to set a particular expiry time relative to now.
cancel	Cancel any asynchronous operations that are waiting on the timer.
cancel_one	Cancels one asynchronous operation that is waiting on the timer.
expires_at	Get the timer's expiry time as an absolute time. Set the timer's expiry time as an absolute time.
expires_from_now	Get the timer's expiry time relative to now. Set the timer's expiry time relative to now.
get_io_service	Get the io_service associated with the object.
wait	Perform a blocking wait on the timer.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_deadline_timer` class template provides the ability to perform a blocking or asynchronous wait for a timer to expire.

A deadline timer is always in one of two states: "expired" or "not expired". If the `wait()` or `async_wait()` function is called on an expired timer, the wait operation will complete immediately.

Most applications will use the `deadline_timer` typedef.

Thread Safety

Distinct objects: Safe.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Shared objects: Unsafe.

Examples

Performing a blocking wait:

```
// Construct a timer without setting an expiry time.
boost::asio::deadline_timer timer(io_service);

// Set an expiry time relative to now.
timer.expires_from_now(boost::posix_time::seconds(5));

// Wait for the timer to expire.
timer.wait();
```

Performing an asynchronous wait:

```
void handler(const boost::system::error_code& error)
{
  if (!error)
  {
    // Timer expired.
  }
}

...

// Construct a timer with an absolute expiry time.
boost::asio::deadline_timer timer(io_service,
    boost::posix_time::time_from_string("2005-12-07 23:59:59.000"));

// Start an asynchronous wait.
timer.async_wait(handler);
```

Changing an active deadline_timer's expiry time

Changing the expiry time of a timer while there are pending asynchronous waits causes those wait operations to be cancelled. To ensure that the action associated with the timer is performed only once, use something like this: used:

```
void on_some_event()
{
  if (my_timer.expires_from_now(seconds(5)) > 0)
  {
    // We managed to cancel the timer. Start new asynchronous wait.
    my_timer.async_wait(on_timeout);
  }
  else
  {
    // Too late, timer has already expired!
  }
}

void on_timeout(const boost::system::error_code& e)
{
  if (e != boost::asio::error::operation_aborted)
  {
    // Timer was not cancelled, take necessary action.
  }
}
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- The `boost::asio::basic_deadline_timer::expires_from_now()` function cancels any pending asynchronous waits, and returns the number of asynchronous waits that were cancelled. If it returns 0 then you were too late and the wait handler has already been executed, or will soon be executed. If it returns 1 then the wait handler was successfully cancelled.
- If a wait handler is cancelled, the boost::system::error_code passed to it contains the value `boost::asio::error::operation_aborted`.

Requirements

Header: `boost/asio/deadline_timer.hpp`

Convenience header: `boost/asio.hpp`

deadline_timer_service

Default service implementation for a timer.

```
template<
    typename TimeType,
    typename TimeTraits = boost::asio::time_traits<TimeType>>
class deadline_timer_service :
  public io_service::service
```

Types

Name	Description
duration_type	The duration type.
implementation_type	The implementation type of the deadline timer.
time_type	The time type.
traits_type	The time traits type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
async_wait	
cancel	Cancel any asynchronous wait operations associated with the timer.
cancel_one	Cancels one asynchronous wait operation associated with the timer.
construct	Construct a new timer implementation.
deadline_timer_service	Construct a new timer service for the specified io_service.
destroy	Destroy a timer implementation.
expires_at	Get the expiry time for the timer as an absolute time. Set the expiry time for the timer as an absolute time.
expires_from_now	Get the expiry time for the timer relative to now. Set the expiry time for the timer relative to now.
get_io_service	Get the io_service object that owns the service.
wait	

Data Members

Name	Description
id	The unique service identifier.

Requirements

Header: `boost/asio/deadline_timer_service.hpp`

Convenience header: `boost/asio.hpp`

deadline_timer_service::async_wait

```
template<
    typename WaitHandler>
void async_wait(
    implementation_type & impl,
    WaitHandler handler);
```

deadline_timer_service::cancel

Cancel any asynchronous wait operations associated with the timer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t cancel(
    implementation_type & impl,
    boost::system::error_code & ec);
```

deadline_timer_service::cancel_one

Cancels one asynchronous wait operation associated with the timer.

```
std::size_t cancel_one(
    implementation_type & impl,
    boost::system::error_code & ec);
```

deadline_timer_service::construct

Construct a new timer implementation.

```
void construct(
    implementation_type & impl);
```

deadline_timer_service::deadline_timer_service

Construct a new timer service for the specified `io_service`.

```
deadline_timer_service(
    boost::asio::io_service & io_service);
```

deadline_timer_service::destroy

Destroy a timer implementation.

```
void destroy(
    implementation_type & impl);
```

deadline_timer_service::duration_type

The duration type.

```
typedef traits_type::duration_type duration_type;
```

Requirements

Header: `boost/asio/deadline_timer_service.hpp`

Convenience header: `boost/asio.hpp`

deadline_timer_service::expires_at

Get the expiry time for the timer as an absolute time.

```
time_type expires_at(
    const implementation_type & impl) const;
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Set the expiry time for the timer as an absolute time.

```
std::size_t expires_at(
    implementation_type & impl,
    const time_type & expiry_time,
    boost::system::error_code & ec);
  » more...
```

deadline_timer_service::expires_at (1 of 2 overloads)

Get the expiry time for the timer as an absolute time.

```
time_type expires_at(
    const implementation_type & impl) const;
```

deadline_timer_service::expires_at (2 of 2 overloads)

Set the expiry time for the timer as an absolute time.

```
std::size_t expires_at(
    implementation_type & impl,
    const time_type & expiry_time,
    boost::system::error_code & ec);
```

deadline_timer_service::expires_from_now

Get the expiry time for the timer relative to now.

```
duration_type expires_from_now(
    const implementation_type & impl) const;
  » more...
```

Set the expiry time for the timer relative to now.

```
std::size_t expires_from_now(
    implementation_type & impl,
    const duration_type & expiry_time,
    boost::system::error_code & ec);
  » more...
```

deadline_timer_service::expires_from_now (1 of 2 overloads)

Get the expiry time for the timer relative to now.

```
duration_type expires_from_now(
    const implementation_type & impl) const;
```

deadline_timer_service::expires_from_now (2 of 2 overloads)

Set the expiry time for the timer relative to now.

```
std::size_t expires_from_now(
    implementation_type & impl,
    const duration_type & expiry_time,
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

deadline_timer_service::get_io_service

Inherited from io_service.

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

deadline_timer_service::id

The unique service identifier.

```
static boost::asio::io_service::id id;
```

deadline_timer_service::implementation_type

The implementation type of the deadline timer.

```
typedef implementation_defined implementation_type;
```

Requirements

Header: `boost/asio/deadline_timer_service.hpp`

Convenience header: `boost/asio.hpp`

deadline_timer_service::time_type

The time type.

```
typedef traits_type::time_type time_type;
```

Requirements

Header: `boost/asio/deadline_timer_service.hpp`

Convenience header: `boost/asio.hpp`

deadline_timer_service::traits_type

The time traits type.

```
typedef TimeTraits traits_type;
```

Requirements

Header: `boost/asio/deadline_timer_service.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

deadline_timer_service::wait

```
void wait(
    implementation_type & impl,
    boost::system::error_code & ec);
```

error::addrinfo_category

```
static const boost::system::error_category & addrinfo_category = boost::asio::error::get_ad↵
drinfo_category();
```

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

error::addrinfo_errors

```
enum addrinfo_errors
```

Values

service_not_found	The service is not supported for the given socket type.
socket_type_not_supported	The socket type is not supported.

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

error::basic_errors

```
enum basic_errors
```

Values

access_denied	Permission denied.
address_family_not_supported	Address family not supported by protocol.
address_in_use	Address already in use.
already_connected	Transport endpoint is already connected.
already_started	Operation already in progress.
broken_pipe	Broken pipe.
connection_aborted	A connection has been aborted.
connection_refused	Connection refused.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

connection_reset	Connection reset by peer.
bad_descriptor	Bad file descriptor.
fault	Bad address.
host_unreachable	No route to host.
in_progress	Operation now in progress.
interrupted	Interrupted system call.
invalid_argument	Invalid argument.
message_size	Message too long.
name_too_long	The name was too long.
network_down	Network is down.
network_reset	Network dropped connection on reset.
network_unreachable	Network is unreachable.
no_descriptors	Too many open files.
no_buffer_space	No buffer space available.
no_memory	Cannot allocate memory.
no_permission	Operation not permitted.
no_protocol_option	Protocol not available.
not_connected	Transport endpoint is not connected.
not_socket	Socket operation on non-socket.
operation_aborted	Operation cancelled.
operation_not_supported	Operation not supported.
shut_down	Cannot send after transport endpoint shutdown.
timed_out	Connection timed out.
try_again	Resource temporarily unavailable.
would_block	The socket is marked non-blocking and the requested operation would block.

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

error::get_addrinfo_category

```
const boost::system::error_category & get_addrinfo_category();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

error::get_misc_category

```
const boost::system::error_category & get_misc_category();
```

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

error::get_netdb_category

```
const boost::system::error_category & get_netdb_category();
```

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

error::get_ssl_category

```
const boost::system::error_category & get_ssl_category();
```

Requirements

Header: `boost/asio/ssl/error.hpp`

Convenience header: `boost/asio/ssl.hpp`

error::get_system_category

```
const boost::system::error_category & get_system_category();
```

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

error::make_error_code

```
boost::system::error_code make_error_code(
    basic_errors e);
  » more...

boost::system::error_code make_error_code(
    netdb_errors e);
  » more...

boost::system::error_code make_error_code(
    addrinfo_errors e);
  » more...

boost::system::error_code make_error_code(
    misc_errors e);
  » more...

boost::system::error_code make_error_code(
    ssl_errors e);
  » more...
```

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

error::make_error_code (1 of 5 overloads)

```
boost::system::error_code make_error_code(
    basic_errors e);
```

error::make_error_code (2 of 5 overloads)

```
boost::system::error_code make_error_code(
    netdb_errors e);
```

error::make_error_code (3 of 5 overloads)

```
boost::system::error_code make_error_code(
    addrinfo_errors e);
```

error::make_error_code (4 of 5 overloads)

```
boost::system::error_code make_error_code(
    misc_errors e);
```

error::make_error_code (5 of 5 overloads)

```
boost::system::error_code make_error_code(
    ssl_errors e);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

error::misc_category

```
static const boost::system::error_category & misc_category = boost::asio::error::get_misc_cat↵
egory();
```

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

error::misc_errors

```
enum misc_errors
```

Values

already_open	Already open.
eof	End of file or stream.
not_found	Element not found.
fd_set_failure	The descriptor cannot fit into the select system call's fd_set.

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

error::netdb_category

```
static const boost::system::error_category & netdb_category = boost::asio::error::get_netdb_cat↵
egory();
```

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

error::netdb_errors

```
enum netdb_errors
```

Values

host_not_found	Host not found (authoritative).
host_not_found_try_again	Host not found (non-authoritative).
no_data	The query is valid but does not have associated address data.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

no_recovery	A non-recoverable error occurred.

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

error::ssl_category

```
static const boost::system::error_category & ssl_category = boost::asio::error::get_ssl_category();
```

Requirements

Header: `boost/asio/ssl/error.hpp`

Convenience header: `boost/asio/ssl.hpp`

error::ssl_errors

```
enum ssl_errors
```

Requirements

Header: `boost/asio/ssl/error.hpp`

Convenience header: `boost/asio/ssl.hpp`

error::system_category

```
static const boost::system::error_category & system_category = boost::asio::error::get_system_cat↵
egory();
```

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

has_service

```
template<
    typename Service>
bool has_service(
    io_service & ios);
```

This function is used to determine whether the `io_service` contains a service object corresponding to the given service type.

Parameters

ios	The `io_service` object that owns the service.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Return Value

A boolean indicating whether the `io_service` contains the service.

Requirements

Header: `boost/asio/io_service.hpp`

Convenience header: `boost/asio.hpp`

high_resolution_timer

Typedef for a timer based on the high resolution clock.

```
typedef basic_waitable_timer< chrono::high_resolution_clock > high_resolution_timer;
```

Types

Name	Description
clock_type	The clock type.
duration	The duration type of the clock.
implementation_type	The underlying implementation type of I/O object.
service_type	The type of the service that will be used to provide I/O operations.
time_point	The time point type of the clock.
traits_type	The wait traits type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
async_wait	Start an asynchronous wait on the timer.
basic_waitable_timer	Constructor. Constructor to set a particular expiry time as an absolute time. Constructor to set a particular expiry time relative to now.
cancel	Cancel any asynchronous operations that are waiting on the timer.
cancel_one	Cancels one asynchronous operation that is waiting on the timer.
expires_at	Get the timer's expiry time as an absolute time. Set the timer's expiry time as an absolute time.
expires_from_now	Get the timer's expiry time relative to now. Set the timer's expiry time relative to now.
get_io_service	Get the io_service associated with the object.
wait	Perform a blocking wait on the timer.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_waitable_timer` class template provides the ability to perform a blocking or asynchronous wait for a timer to expire.

A waitable timer is always in one of two states: "expired" or "not expired". If the `wait()` or `async_wait()` function is called on an expired timer, the wait operation will complete immediately.

Most applications will use the boost::asio::waitable_timer typedef.

Remarks

This waitable timer functionality is for use with the C++11 standard library's `<chrono>` facility, or with the Boost.Chrono library.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Examples

Performing a blocking wait:

```
// Construct a timer without setting an expiry time.
boost::asio::waitable_timer timer(io_service);

// Set an expiry time relative to now.
timer.expires_from_now(boost::posix_time::seconds(5));

// Wait for the timer to expire.
timer.wait();
```

Performing an asynchronous wait:

```
void handler(const boost::system::error_code& error)
{
  if (!error)
  {
    // Timer expired.
  }
}

...

// Construct a timer with an absolute expiry time.
boost::asio::waitable_timer timer(io_service,
    boost::posix_time::time_from_string("2005-12-07 23:59:59.000"));

// Start an asynchronous wait.
timer.async_wait(handler);
```

Changing an active waitable_timer's expiry time

Changing the expiry time of a timer while there are pending asynchronous waits causes those wait operations to be cancelled. To ensure that the action associated with the timer is performed only once, use something like this: used:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void on_some_event()
{
  if (my_timer.expires_from_now(seconds(5)) > 0)
  {
    // We managed to cancel the timer. Start new asynchronous wait.
    my_timer.async_wait(on_timeout);
  }
  else
  {
    // Too late, timer has already expired!
  }
}

void on_timeout(const boost::system::error_code& e)
{
  if (e != boost::asio::error::operation_aborted)
  {
    // Timer was not cancelled, take necessary action.
  }
}
```

- The `boost::asio::basic_waitable_timer::expires_from_now()` function cancels any pending asynchronous waits, and returns the number of asynchronous waits that were cancelled. If it returns 0 then you were too late and the wait handler has already been executed, or will soon be executed. If it returns 1 then the wait handler was successfully cancelled.
- If a wait handler is cancelled, the boost::system::error_code passed to it contains the value `boost::asio::error::operation_aborted`.

This typedef uses the C++11 `<chrono>` standard library facility, if available. Otherwise, it may use the Boost.Chrono library. To explicitly utilise Boost.Chrono, use the `basic_waitable_timer` template directly:

```
typedef basic_waitable_timer<boost::chrono::high_resolution_clock> timer;
```

Requirements

Header: `boost/asio/high_resolution_timer.hpp`

Convenience header: None

invalid_service_owner

Exception thrown when trying to add a service object to an `io_service` where the service has a different owner.

```
class invalid_service_owner
```

Member Functions

Name	Description
invalid_service_owner	

Requirements

Header: `boost/asio/io_service.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

invalid_service_owner::invalid_service_owner

```
invalid_service_owner();
```

io_service

Provides core I/O functionality.

```
class io_service :
  noncopyable
```

Types

Name	Description
id	Class used to uniquely identify a service.
service	Base class for all io_service services.
strand	Provides serialised handler execution.
work	Class to inform the io_service when it has work to do.
fork_event	Fork-related event notifications.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
dispatch	Request the io_service to invoke the given handler.
io_service	Constructor.
notify_fork	Notify the io_service of a fork-related event.
poll	Run the io_service object's event processing loop to execute ready handlers.
poll_one	Run the io_service object's event processing loop to execute one ready handler.
post	Request the io_service to invoke the given handler and return immediately.
reset	Reset the io_service in preparation for a subsequent run() invocation.
run	Run the io_service object's event processing loop.
run_one	Run the io_service object's event processing loop to execute at most one handler.
stop	Stop the io_service object's event processing loop.
stopped	Determine whether the io_service object has been stopped.
wrap	Create a new handler that automatically dispatches the wrapped handler on the io_service.
~io_service	Destructor.

Friends

Name	Description
add_service	Add a service object to the io_service.
has_service	Determine if an io_service contains a specified service type.
use_service	Obtain the service object corresponding to the given type.

The `io_service` class provides the core I/O functionality for users of the asynchronous I/O objects, including:

- `boost::asio::ip::tcp::socket`
- `boost::asio::ip::tcp::acceptor`
- `boost::asio::ip::udp::socket`
- `deadline_timer`.

The `io_service` class also includes facilities intended for developers of custom asynchronous services.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Thread Safety

Distinct objects: Safe.

Shared objects: Safe, with the specific exceptions of the `reset()` and `notify_fork()` functions. Calling `reset()` while there are unfinished `run()`, `run_one()`, `poll()` or `poll_one()` calls results in undefined behaviour. The `notify_fork()` function should not be called while any `io_service` function, or any function on an I/O object that is associated with the `io_service`, is being called in another thread.

Synchronous and asynchronous operations

Synchronous operations on I/O objects implicitly run the `io_service` object for an individual operation. The `io_service` functions `run()`, `run_one()`, `poll()` or `poll_one()` must be called for the `io_service` to perform asynchronous operations on behalf of a C++ program. Notification that an asynchronous operation has completed is delivered by invocation of the associated handler. Handlers are invoked only by a thread that is currently calling any overload of `run()`, `run_one()`, `poll()` or `poll_one()` for the `io_service`.

Effect of exceptions thrown from handlers

If an exception is thrown from a handler, the exception is allowed to propagate through the throwing thread's invocation of `run()`, `run_one()`, `poll()` or `poll_one()`. No other threads that are calling any of these functions are affected. It is then the responsibility of the application to catch the exception.

After the exception has been caught, the `run()`, `run_one()`, `poll()` or `poll_one()` call may be restarted **without** the need for an intervening call to `reset()`. This allows the thread to rejoin the `io_service` object's thread pool without impacting any other threads in the pool.

For example:

```
boost::asio::io_service io_service;
...
for (;;)
{
  try
  {
    io_service.run();
    break; // run() exited normally
  }
  catch (my_exception& e)
  {
    // Deal with exception as appropriate.
  }
}
```

Stopping the io_service from running out of work

Some applications may need to prevent an `io_service` object's `run()` call from returning when there is no more work to do. For example, the `io_service` may be being run in a background thread that is launched prior to the application's asynchronous operations. The `run()` call may be kept running by creating an object of type `io_service::work`:

```
boost::asio::io_service io_service;
boost::asio::io_service::work work(io_service);
...
```

To effect a shutdown, the application will then need to call the `io_service` object's `stop()` member function. This will cause the `io_service` `run()` call to return as soon as possible, abandoning unfinished operations and without permitting ready handlers to be dispatched.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Alternatively, if the application requires that all operations and handlers be allowed to finish normally, the work object may be explicitly destroyed.

```
boost::asio::io_service io_service;
auto_ptr<boost::asio::io_service::work> work(
    new boost::asio::io_service::work(io_service));
...
work.reset(); // Allow run() to exit.
```

The io_service class and I/O services

Class `io_service` implements an extensible, type-safe, polymorphic set of I/O services, indexed by service type. An object of class `io_service` must be initialised before I/O objects such as sockets, resolvers and timers can be used. These I/O objects are distinguished by having constructors that accept an `io_service&` parameter.

I/O services exist to manage the logical interface to the operating system on behalf of the I/O objects. In particular, there are resources that are shared across a class of I/O objects. For example, timers may be implemented in terms of a single timer queue. The I/O services manage these shared resources.

Access to the services of an `io_service` is via three function templates, `use_service()`, `add_service()` and `has_service()`.

In a call to `use_service<Service>()`, the type argument chooses a service, making available all members of the named type. If `Service` is not present in an `io_service`, an object of type `Service` is created and added to the `io_service`. A C++ program can check if an `io_service` implements a particular service with the function template `has_service<Service>()`.

Service objects may be explicitly added to an `io_service` using the function template `add_service<Service>()`. If the `Service` is already present, the `service_already_exists` exception is thrown. If the owner of the service is not the same object as the `io_service` parameter, the `invalid_service_owner` exception is thrown.

Once a service reference is obtained from an `io_service` object by calling `use_service()`, that reference remains usable as long as the owning `io_service` object exists.

All I/O service implementations have `io_service::service` as a public base class. Custom I/O services may be implemented by deriving from this class and then added to an `io_service` using the facilities described above.

Requirements

Header: `boost/asio/io_service.hpp`

Convenience header: `boost/asio.hpp`

io_service::add_service

Add a service object to the `io_service`.

```
template<
    typename Service>
friend void add_service(
    io_service & ios,
    Service * svc);
```

This function is used to add a service to the `io_service`.

Parameters

ios — The `io_service` object that owns the service.

svc — The service object. On success, ownership of the service object is transferred to the `io_service`. When the `io_service` object is destroyed, it will destroy the service object by performing:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
delete static_cast<io_service::service*>(svc)
```

Exceptions

boost::asio::service_already_exists	Thrown if a service of the given type is already present in the `io_service`.
boost::asio::invalid_service_owner	Thrown if the service's owning `io_service` is not the `io_service` object specified by the ios parameter.

Requirements

Header: `boost/asio/io_service.hpp`

Convenience header: `boost/asio.hpp`

io_service::dispatch

Request the `io_service` to invoke the given handler.

```
template<
    typename CompletionHandler>
void dispatch(
    CompletionHandler handler);
```

This function is used to ask the `io_service` to execute the given handler.

The `io_service` guarantees that the handler will only be called in a thread in which the `run()`, `run_one()`, `poll()` or `poll_one()` member functions is currently being invoked. The handler may be executed inside this function if the guarantee can be met.

Parameters

handler	The handler to be called. The `io_service` will make a copy of the handler object as required. The function signature of the handler must be: `void handler();`

Remarks

This function throws an exception only if:

- the handler's `asio_handler_allocate` function; or
- the handler's copy constructor

throws an exception.

io_service::fork_event

Fork-related event notifications.

```
enum fork_event
```

Values

fork_prepare	Notify the io_service that the process is about to fork.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

fork_parent	Notify the io_service that the process has forked and is the parent.
fork_child	Notify the io_service that the process has forked and is the child.

io_service::has_service

Determine if an `io_service` contains a specified service type.

```
template<
    typename Service>
friend bool has_service(
    io_service & ios);
```

This function is used to determine whether the `io_service` contains a service object corresponding to the given service type.

Parameters

ios	The `io_service` object that owns the service.

Return Value

A boolean indicating whether the `io_service` contains the service.

Requirements

Header: `boost/asio/io_service.hpp`

Convenience header: `boost/asio.hpp`

io_service::io_service

Constructor.

```
io_service();
  » more...

explicit io_service(
    std::size_t concurrency_hint);
  » more...
```

io_service::io_service (1 of 2 overloads)

Constructor.

```
io_service();
```

io_service::io_service (2 of 2 overloads)

Constructor.

```
io_service(
    std::size_t concurrency_hint);
```

Construct with a hint about the required level of concurrency.

Parameters

concurrency_hint	A suggestion to the implementation on how many threads it should allow to run simultaneously.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

io_service::notify_fork

Notify the io_service of a fork-related event.

```
void notify_fork(
    boost::asio::io_service::fork_event event);
```

This function is used to inform the io_service that the process is about to fork, or has just forked. This allows the io_service, and the services it contains, to perform any necessary housekeeping to ensure correct operation following a fork.

This function must not be called while any other io_service function, or any function on an I/O object associated with the io_service, is being called in another thread. It is, however, safe to call this function from within a completion handler, provided no other thread is accessing the io_service.

Parameters

event A fork-related event.

Exceptions

boost::system::system_error Thrown on failure. If the notification fails the io_service object should no longer be used and should be destroyed.

Example

The following code illustrates how to incorporate the `notify_fork()` function:

```
my_io_service.notify_fork(boost::asio::io_service::fork_prepare);
if (fork() == 0)
{
  // This is the child process.
  my_io_service.notify_fork(boost::asio::io_service::fork_child);
}
else
{
  // This is the parent process.
  my_io_service.notify_fork(boost::asio::io_service::fork_parent);
}
```

Remarks

For each service object `svc` in the io_service set, performs `svc->fork_service();`. When processing the fork_prepare event, services are visited in reverse order of the beginning of service object lifetime. Otherwise, services are visited in order of the beginning of service object lifetime.

io_service::poll

Run the io_service object's event processing loop to execute ready handlers.

```
std::size_t poll();
  » more...

std::size_t poll(
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

io_service::poll (1 of 2 overloads)

Run the `io_service` object's event processing loop to execute ready handlers.

```
std::size_t poll();
```

The `poll()` function runs handlers that are ready to run, without blocking, until the `io_service` has been stopped or there are no more ready handlers.

Return Value

The number of handlers that were executed.

Exceptions

boost::system::system_error Thrown on failure.

io_service::poll (2 of 2 overloads)

Run the `io_service` object's event processing loop to execute ready handlers.

```
std::size_t poll(
    boost::system::error_code & ec);
```

The `poll()` function runs handlers that are ready to run, without blocking, until the `io_service` has been stopped or there are no more ready handlers.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

The number of handlers that were executed.

io_service::poll_one

Run the `io_service` object's event processing loop to execute one ready handler.

```
std::size_t poll_one();
  » more...

std::size_t poll_one(
    boost::system::error_code & ec);
  » more...
```

io_service::poll_one (1 of 2 overloads)

Run the `io_service` object's event processing loop to execute one ready handler.

```
std::size_t poll_one();
```

The `poll_one()` function runs at most one handler that is ready to run, without blocking.

Return Value

The number of handlers that were executed.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

io_service::poll_one (2 of 2 overloads)

Run the `io_service` object's event processing loop to execute one ready handler.

```
std::size_t poll_one(
    boost::system::error_code & ec);
```

The `poll_one()` function runs at most one handler that is ready to run, without blocking.

Parameters

ec	Set to indicate what error occurred, if any.

Return Value

The number of handlers that were executed.

io_service::post

Request the `io_service` to invoke the given handler and return immediately.

```
template<
    typename CompletionHandler>
void post(
    CompletionHandler handler);
```

This function is used to ask the `io_service` to execute the given handler, but without allowing the `io_service` to call the handler from inside this function.

The `io_service` guarantees that the handler will only be called in a thread in which the `run()`, `run_one()`, `poll()` or `poll_one()` member functions is currently being invoked.

Parameters

handler	The handler to be called. The `io_service` will make a copy of the handler object as required. The function signature of the handler must be: `void handler();`

Remarks

This function throws an exception only if:

- the handler's `asio_handler_allocate` function; or
- the handler's copy constructor

throws an exception.

io_service::reset

Reset the `io_service` in preparation for a subsequent `run()` invocation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void reset();
```

This function must be called prior to any second or later set of invocations of the `run()`, `run_one()`, `poll()` or `poll_one()` functions when a previous invocation of these functions returned due to the `io_service` being stopped or running out of work. After a call to `reset()`, the `io_service` object's `stopped()` function will return `false`.

This function must not be called while there are any unfinished calls to the `run()`, `run_one()`, `poll()` or `poll_one()` functions.

io_service::run

Run the `io_service` object's event processing loop.

```
std::size_t run();
  » more...

std::size_t run(
    boost::system::error_code & ec);
  » more...
```

io_service::run (1 of 2 overloads)

Run the `io_service` object's event processing loop.

```
std::size_t run();
```

The `run()` function blocks until all work has finished and there are no more handlers to be dispatched, or until the `io_service` has been stopped.

Multiple threads may call the `run()` function to set up a pool of threads from which the `io_service` may execute handlers. All threads that are waiting in the pool are equivalent and the `io_service` may choose any one of them to invoke a handler.

A normal exit from the `run()` function implies that the `io_service` object is stopped (the `stopped()` function returns `true`). Subsequent calls to `run()`, `run_one()`, `poll()` or `poll_one()` will return immediately unless there is a prior call to `reset()`.

Return Value

The number of handlers that were executed.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

The `run()` function must not be called from a thread that is currently calling one of `run()`, `run_one()`, `poll()` or `poll_one()` on the same `io_service` object.

The `poll()` function may also be used to dispatch ready handlers, but without blocking.

io_service::run (2 of 2 overloads)

Run the `io_service` object's event processing loop.

```
std::size_t run(
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The `run()` function blocks until all work has finished and there are no more handlers to be dispatched, or until the `io_service` has been stopped.

Multiple threads may call the `run()` function to set up a pool of threads from which the `io_service` may execute handlers. All threads that are waiting in the pool are equivalent and the `io_service` may choose any one of them to invoke a handler.

A normal exit from the `run()` function implies that the `io_service` object is stopped (the `stopped()` function returns `true`). Subsequent calls to `run()`, `run_one()`, `poll()` or `poll_one()` will return immediately unless there is a prior call to `reset()`.

Parameters

ec Set to indicate what error occurred, if any.

Return Value

The number of handlers that were executed.

Remarks

The `run()` function must not be called from a thread that is currently calling one of `run()`, `run_one()`, `poll()` or `poll_one()` on the same `io_service` object.

The `poll()` function may also be used to dispatch ready handlers, but without blocking.

io_service::run_one

Run the `io_service` object's event processing loop to execute at most one handler.

```
std::size_t run_one();
  » more...

std::size_t run_one(
    boost::system::error_code & ec);
  » more...
```

io_service::run_one (1 of 2 overloads)

Run the `io_service` object's event processing loop to execute at most one handler.

```
std::size_t run_one();
```

The `run_one()` function blocks until one handler has been dispatched, or until the `io_service` has been stopped.

Return Value

The number of handlers that were executed. A zero return value implies that the `io_service` object is stopped (the `stopped()` function returns `true`). Subsequent calls to `run()`, `run_one()`, `poll()` or `poll_one()` will return immediately unless there is a prior call to `reset()`.

Exceptions

boost::system::system_error Thrown on failure.

io_service::run_one (2 of 2 overloads)

Run the `io_service` object's event processing loop to execute at most one handler.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t run_one(
    boost::system::error_code & ec);
```

The `run_one()` function blocks until one handler has been dispatched, or until the `io_service` has been stopped.

Return Value

The number of handlers that were executed. A zero return value implies that the `io_service` object is stopped (the `stopped()` function returns `true`). Subsequent calls to `run()`, `run_one()`, `poll()` or `poll_one()` will return immediately unless there is a prior call to `reset()`.

The number of handlers that were executed.

io_service::stop

Stop the `io_service` object's event processing loop.

```
void stop();
```

This function does not block, but instead simply signals the `io_service` to stop. All invocations of its `run()` or `run_one()` member functions should return as soon as possible. Subsequent calls to `run()`, `run_one()`, `poll()` or `poll_one()` will return immediately until `reset()` is called.

io_service::stopped

Determine whether the `io_service` object has been stopped.

```
bool stopped() const;
```

This function is used to determine whether an `io_service` object has been stopped, either through an explicit call to `stop()`, or due to running out of work. When an `io_service` object is stopped, calls to `run()`, `run_one()`, `poll()` or `poll_one()` will return immediately without invoking any handlers.

Return Value

`true` if the `io_service` object is stopped, otherwise `false`.

io_service::use_service

Obtain the service object corresponding to the given type.

```
template<
    typename Service>
friend Service & use_service(
    io_service & ios);
```

This function is used to locate a service object that corresponds to the given service type. If there is no existing implementation of the service, then the `io_service` will create a new instance of the service.

Parameters

ios The `io_service` object that owns the service.

Return Value

The service interface implementing the specified service type. Ownership of the service interface is not transferred to the caller.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/io_service.hpp`

Convenience header: `boost/asio.hpp`

io_service::wrap

Create a new handler that automatically dispatches the wrapped handler on the `io_service`.

```
template<
    typename Handler>
unspecified wrap(
    Handler handler);
```

This function is used to create a new handler function object that, when invoked, will automatically pass the wrapped handler to the `io_service` object's dispatch function.

Parameters

handler — The handler to be wrapped. The `io_service` will make a copy of the handler object as required. The function signature of the handler must be:

```
void handler(A1 a1, ... An an);
```

Return Value

A function object that, when invoked, passes the wrapped handler to the `io_service` object's dispatch function. Given a function object with the signature:

```
R f(A1 a1, ... An an);
```

If this function object is passed to the wrap function like so:

```
io_service.wrap(f);
```

then the return value is a function object with the signature

```
void g(A1 a1, ... An an);
```

that, when invoked, executes code equivalent to:

```
io_service.dispatch(boost::bind(f, a1, ... an));
```

io_service::~io_service

Destructor.

```
~io_service();
```

On destruction, the `io_service` performs the following sequence of operations:

- For each service object `svc` in the `io_service` set, in reverse order of the beginning of service object lifetime, performs `svc->shutdown_service()`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- Uninvoked handler objects that were scheduled for deferred invocation on the `io_service`, or any associated strand, are destroyed.
- For each service object `svc` in the `io_service` set, in reverse order of the beginning of service object lifetime, performs `delete static_cast<io_service::service*>(svc)`.

Remarks

The destruction sequence described above permits programs to simplify their resource management by using `shared_ptr<>`. Where an object's lifetime is tied to the lifetime of a connection (or some other sequence of asynchronous operations), a `shared_ptr` to the object would be bound into the handlers for all asynchronous operations associated with it. This works as follows:

- When a single connection ends, all associated asynchronous operations complete. The corresponding handler objects are destroyed, and all `shared_ptr` references to the objects are destroyed.
- To shut down the whole program, the `io_service` function `stop()` is called to terminate any `run()` calls as soon as possible. The `io_service` destructor defined above destroys all handlers, causing all `shared_ptr` references to all connection objects to be destroyed.

io_service::id

Class used to uniquely identify a service.

```
class id :
  noncopyable
```

Member Functions

Name	Description
id	Constructor.

Requirements

Header: `boost/asio/io_service.hpp`

Convenience header: `boost/asio.hpp`

io_service::id::id

Constructor.

```
id();
```

io_service::service

Base class for all `io_service` services.

```
class service :
  noncopyable
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
get_io_service	Get the io_service object that owns the service.

Protected Member Functions

Name	Description
service	Constructor.
~service	Destructor.

Private Member Functions

Name	Description
fork_service	Handle notification of a fork-related event to perform any necessary housekeeping.
shutdown_service	Destroy all user-defined handler objects owned by the service.

Requirements

Header: `boost/asio/io_service.hpp`

Convenience header: `boost/asio.hpp`

io_service::service::get_io_service

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

io_service::service::service

Constructor.

```
service(
    boost::asio::io_service & owner);
```

Parameters

owner The `io_service` object that owns the service.

io_service::service::~service

Destructor.

```
virtual ~service();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

io_service::service::fork_service

Handle notification of a fork-related event to perform any necessary housekeeping.

```
virtual void fork_service(
    boost::asio::io_service::fork_event event);
```

This function is not a pure virtual so that services only have to implement it if necessary. The default implementation does nothing.

io_service::service::shutdown_service

Destroy all user-defined handler objects owned by the service.

```
void shutdown_service();
```

io_service::strand

Provides serialised handler execution.

```
class strand
```

Member Functions

Name	Description
dispatch	Request the strand to invoke the given handler.
get_io_service	Get the io_service associated with the strand.
post	Request the strand to invoke the given handler and return immediately.
strand	Constructor.
wrap	Create a new handler that automatically dispatches the wrapped handler on the strand.
~strand	Destructor.

The `io_service::strand` class provides the ability to post and dispatch handlers with the guarantee that none of those handlers will execute concurrently.

Order of handler invocation

Given:

- a strand object `s`
- an object `a` meeting completion handler requirements
- an object `a1` which is an arbitrary copy of `a` made by the implementation
- an object `b` meeting completion handler requirements
- an object `b1` which is an arbitrary copy of `b` made by the implementation

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

if any of the following conditions are true:

- `s.post(a)` happens-before `s.post(b)`
- `s.post(a)` happens-before `s.dispatch(b)`, where the latter is performed outside the strand
- `s.dispatch(a)` happens-before `s.post(b)`, where the former is performed outside the strand
- `s.dispatch(a)` happens-before `s.dispatch(b)`, where both are performed outside the strand

then `asio_handler_invoke(a1, &a1)` happens-before `asio_handler_invoke(b1, &b1)`.

Note that in the following case:

```
async_op_1(..., s.wrap(a));
async_op_2(..., s.wrap(b));
```

the completion of the first async operation will perform `s.dispatch(a)`, and the second will perform `s.dispatch(b)`, but the order in which those are performed is unspecified. That is, you cannot state whether one happens-before the other. Therefore none of the above conditions are met and no ordering guarantee is made.

Thread Safety

Distinct objects: Safe.

Shared objects: Safe.

Requirements

Header: `boost/asio/strand.hpp`

Convenience header: `boost/asio.hpp`

io_service::strand::dispatch

Request the strand to invoke the given handler.

```
template<
    typename CompletionHandler>
void dispatch(
    CompletionHandler handler);
```

This function is used to ask the strand to execute the given handler.

The strand object guarantees that handlers posted or dispatched through the strand will not be executed concurrently. The handler may be executed inside this function if the guarantee can be met. If this function is called from within a handler that was posted or dispatched through the same strand, then the new handler will be executed immediately.

The strand's guarantee is in addition to the guarantee provided by the underlying `io_service`. The `io_service` guarantees that the handler will only be called in a thread in which the io_service's run member function is currently being invoked.

Parameters

handler — The handler to be called. The strand will make a copy of the handler object as required. The function signature of the handler must be:

```
void handler();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

io_service::strand::get_io_service

Get the `io_service` associated with the strand.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the strand uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the strand will use to dispatch handlers. Ownership is not transferred to the caller.

io_service::strand::post

Request the strand to invoke the given handler and return immediately.

```
template<
    typename CompletionHandler>
void post(
    CompletionHandler handler);
```

This function is used to ask the strand to execute the given handler, but without allowing the strand to call the handler from inside this function.

The strand object guarantees that handlers posted or dispatched through the strand will not be executed concurrently. The strand's guarantee is in addition to the guarantee provided by the underlying `io_service`. The `io_service` guarantees that the handler will only be called in a thread in which the io_service's run member function is currently being invoked.

Parameters

handler — The handler to be called. The strand will make a copy of the handler object as required. The function signature of the handler must be:

```
void handler();
```

io_service::strand::strand

Constructor.

```
strand(
    boost::asio::io_service & io_service);
```

Constructs the strand.

Parameters

io_service — The `io_service` object that the strand will use to dispatch handlers that are ready to be run.

io_service::strand::wrap

Create a new handler that automatically dispatches the wrapped handler on the strand.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename Handler>
unspecified wrap(
    Handler handler);
```

This function is used to create a new handler function object that, when invoked, will automatically pass the wrapped handler to the strand's dispatch function.

Parameters

handler — The handler to be wrapped. The strand will make a copy of the handler object as required. The function signature of the handler must be:

```
void handler(A1 a1, ... An an);
```

Return Value

A function object that, when invoked, passes the wrapped handler to the strand's dispatch function. Given a function object with the signature:

```
R f(A1 a1, ... An an);
```

If this function object is passed to the wrap function like so:

```
strand.wrap(f);
```

then the return value is a function object with the signature

```
void g(A1 a1, ... An an);
```

that, when invoked, executes code equivalent to:

```
strand.dispatch(boost::bind(f, a1, ... an));
```

io_service::strand::~strand

Destructor.

```
~strand();
```

Destroys a strand.

Handlers posted through the strand that have not yet been invoked will still be dispatched in a way that meets the guarantee of non-concurrency.

io_service::work

Class to inform the `io_service` when it has work to do.

```
class work
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
get_io_service	Get the io_service associated with the work.
work	Constructor notifies the io_service that work is starting. Copy constructor notifies the io_service that work is starting.
~work	Destructor notifies the io_service that the work is complete.

The work class is used to inform the `io_service` when work starts and finishes. This ensures that the `io_service` object's `run()` function will not exit while work is underway, and that it does exit when there is no unfinished work remaining.

The work class is copy-constructible so that it may be used as a data member in a handler class. It is not assignable.

Requirements

Header: `boost/asio/io_service.hpp`

Convenience header: `boost/asio.hpp`

io_service::work::get_io_service

Get the `io_service` associated with the work.

```
boost::asio::io_service & get_io_service();
```

io_service::work::work

Constructor notifies the `io_service` that work is starting.

```
explicit work(
    boost::asio::io_service & io_service);
  » more...
```

Copy constructor notifies the `io_service` that work is starting.

```
work(
    const work & other);
  » more...
```

io_service::work::work (1 of 2 overloads)

Constructor notifies the `io_service` that work is starting.

```
work(
    boost::asio::io_service & io_service);
```

The constructor is used to inform the `io_service` that some work has begun. This ensures that the `io_service` object's `run()` function will not exit while the work is underway.

io_service::work::work (2 of 2 overloads)

Copy constructor notifies the `io_service` that work is starting.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
work(
    const work & other);
```

The constructor is used to inform the `io_service` that some work has begun. This ensures that the `io_service` object's `run()` function will not exit while the work is underway.

io_service::work::~work

Destructor notifies the `io_service` that the work is complete.

```
~work();
```

The destructor is used to inform the `io_service` that some work has finished. Once the count of unfinished work reaches zero, the `io_service` object's `run()` function is permitted to exit.

ip::address

Implements version-independent IP addresses.

```
class address
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
address	Default constructor. Construct an address from an IPv4 address. Construct an address from an IPv6 address. Copy constructor.
from_string	Create an address from an IPv4 address string in dotted decimal form, or from an IPv6 address in hexadecimal notation.
is_loopback	Determine whether the address is a loopback address.
is_multicast	Determine whether the address is a multicast address.
is_unspecified	Determine whether the address is unspecified.
is_v4	Get whether the address is an IP version 4 address.
is_v6	Get whether the address is an IP version 6 address.
operator=	Assign from another address. Assign from an IPv4 address. Assign from an IPv6 address.
to_string	Get the address as a string in dotted decimal format.
to_v4	Get the address as an IP version 4 address.
to_v6	Get the address as an IP version 6 address.

Friends

Name	Description
operator!=	Compare two addresses for inequality.
operator<	Compare addresses for ordering.
operator<=	Compare addresses for ordering.
operator==	Compare two addresses for equality.
operator>	Compare addresses for ordering.
operator>=	Compare addresses for ordering.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Related Functions

Name	Description
operator<<	Output an address as a string.

The `ip::address` class provides the ability to use either IP version 4 or version 6 addresses.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/address.hpp`

Convenience header: `boost/asio.hpp`

ip::address::address

Default constructor.

```
address();
  » more...
```

Construct an address from an IPv4 address.

```
address(
    const boost::asio::ip::address_v4 & ipv4_address);
  » more...
```

Construct an address from an IPv6 address.

```
address(
    const boost::asio::ip::address_v6 & ipv6_address);
  » more...
```

Copy constructor.

```
address(
    const address & other);
  » more...
```

ip::address::address (1 of 4 overloads)

Default constructor.

```
address();
```

ip::address::address (2 of 4 overloads)

Construct an address from an IPv4 address.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
address(
    const boost::asio::ip::address_v4 & ipv4_address);
```

ip::address::address (3 of 4 overloads)

Construct an address from an IPv6 address.

```
address(
    const boost::asio::ip::address_v6 & ipv6_address);
```

ip::address::address (4 of 4 overloads)

Copy constructor.

```
address(
    const address & other);
```

ip::address::from_string

Create an address from an IPv4 address string in dotted decimal form, or from an IPv6 address in hexadecimal notation.

```
static address from_string(
    const char * str);
  » more...

static address from_string(
    const char * str,
    boost::system::error_code & ec);
  » more...

static address from_string(
    const std::string & str);
  » more...

static address from_string(
    const std::string & str,
    boost::system::error_code & ec);
  » more...
```

ip::address::from_string (1 of 4 overloads)

Create an address from an IPv4 address string in dotted decimal form, or from an IPv6 address in hexadecimal notation.

```
static address from_string(
    const char * str);
```

ip::address::from_string (2 of 4 overloads)

Create an address from an IPv4 address string in dotted decimal form, or from an IPv6 address in hexadecimal notation.

```
static address from_string(
    const char * str,
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::address::from_string (3 of 4 overloads)

Create an address from an IPv4 address string in dotted decimal form, or from an IPv6 address in hexadecimal notation.

```
static address from_string(
    const std::string & str);
```

ip::address::from_string (4 of 4 overloads)

Create an address from an IPv4 address string in dotted decimal form, or from an IPv6 address in hexadecimal notation.

```
static address from_string(
    const std::string & str,
    boost::system::error_code & ec);
```

ip::address::is_loopback

Determine whether the address is a loopback address.

```
bool is_loopback() const;
```

ip::address::is_multicast

Determine whether the address is a multicast address.

```
bool is_multicast() const;
```

ip::address::is_unspecified

Determine whether the address is unspecified.

```
bool is_unspecified() const;
```

ip::address::is_v4

Get whether the address is an IP version 4 address.

```
bool is_v4() const;
```

ip::address::is_v6

Get whether the address is an IP version 6 address.

```
bool is_v6() const;
```

ip::address::operator!=

Compare two addresses for inequality.

```
friend bool operator!=(
    const address & a1,
    const address & a2);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/ip/address.hpp`

Convenience header: `boost/asio.hpp`

ip::address::operator<

Compare addresses for ordering.

```
friend bool operator<(
    const address & a1,
    const address & a2);
```

Requirements

Header: `boost/asio/ip/address.hpp`

Convenience header: `boost/asio.hpp`

ip::address::operator<<

Output an address as a string.

```
template<
    typename Elem,
    typename Traits>
std::basic_ostream< Elem, Traits > & operator<<(
    std::basic_ostream< Elem, Traits > & os,
    const address & addr);
```

Used to output a human-readable string for a specified address.

Parameters

os The output stream to which the string will be written.

addr The address to be written.

Return Value

The output stream.

ip::address::operator<=

Compare addresses for ordering.

```
friend bool operator<=(
    const address & a1,
    const address & a2);
```

Requirements

Header: `boost/asio/ip/address.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::address::operator=

Assign from another address.

```
address & operator=(
    const address & other);
  » more...
```

Assign from an IPv4 address.

```
address & operator=(
    const boost::asio::ip::address_v4 & ipv4_address);
  » more...
```

Assign from an IPv6 address.

```
address & operator=(
    const boost::asio::ip::address_v6 & ipv6_address);
  » more...
```

ip::address::operator= (1 of 3 overloads)

Assign from another address.

```
address & operator=(
    const address & other);
```

ip::address::operator= (2 of 3 overloads)

Assign from an IPv4 address.

```
address & operator=(
    const boost::asio::ip::address_v4 & ipv4_address);
```

ip::address::operator= (3 of 3 overloads)

Assign from an IPv6 address.

```
address & operator=(
    const boost::asio::ip::address_v6 & ipv6_address);
```

ip::address::operator==

Compare two addresses for equality.

```
friend bool operator==(
    const address & a1,
    const address & a2);
```

Requirements

Header: `boost/asio/ip/address.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::address::operator>

Compare addresses for ordering.

```
friend bool operator>(
    const address & a1,
    const address & a2);
```

Requirements

Header: `boost/asio/ip/address.hpp`

Convenience header: `boost/asio.hpp`

ip::address::operator>=

Compare addresses for ordering.

```
friend bool operator>=(
    const address & a1,
    const address & a2);
```

Requirements

Header: `boost/asio/ip/address.hpp`

Convenience header: `boost/asio.hpp`

ip::address::to_string

Get the address as a string in dotted decimal format.

```
std::string to_string() const;
  » more...

std::string to_string(
    boost::system::error_code & ec) const;
  » more...
```

ip::address::to_string (1 of 2 overloads)

Get the address as a string in dotted decimal format.

```
std::string to_string() const;
```

ip::address::to_string (2 of 2 overloads)

Get the address as a string in dotted decimal format.

```
std::string to_string(
    boost::system::error_code & ec) const;
```

ip::address::to_v4

Get the address as an IP version 4 address.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::address_v4 to_v4() const;
```

ip::address::to_v6

Get the address as an IP version 6 address.

```
boost::asio::ip::address_v6 to_v6() const;
```

ip::address_v4

Implements IP version 4 style addresses.

```
class address_v4
```

Types

Name	Description
bytes_type	The type used to represent an address as an array of bytes.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
address_v4	Default constructor. Construct an address from raw bytes. Construct an address from a unsigned long in host byte order. Copy constructor.
any	Obtain an address object that represents any address.
broadcast	Obtain an address object that represents the broadcast address. Obtain an address object that represents the broadcast address that corresponds to the specified address and netmask.
from_string	Create an address from an IP address string in dotted decimal form.
is_class_a	Determine whether the address is a class A address.
is_class_b	Determine whether the address is a class B address.
is_class_c	Determine whether the address is a class C address.
is_loopback	Determine whether the address is a loopback address.
is_multicast	Determine whether the address is a multicast address.
is_unspecified	Determine whether the address is unspecified.
loopback	Obtain an address object that represents the loopback address.
netmask	Obtain the netmask that corresponds to the address, based on its address class.
operator=	Assign from another address.
to_bytes	Get the address in bytes, in network byte order.
to_string	Get the address as a string in dotted decimal format.
to_ulong	Get the address as an unsigned long in host byte order.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Friends

Name	Description
operator!=	Compare two addresses for inequality.
operator<	Compare addresses for ordering.
operator<=	Compare addresses for ordering.
operator==	Compare two addresses for equality.
operator>	Compare addresses for ordering.
operator>=	Compare addresses for ordering.

Related Functions

Name	Description
operator<<	Output an address as a string.

The `ip::address_v4` class provides the ability to use and manipulate IP version 4 addresses.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/address_v4.hpp`

Convenience header: `boost/asio.hpp`

ip::address_v4::address_v4

Default constructor.

```
address_v4();
  » more...
```

Construct an address from raw bytes.

```
explicit address_v4(
    const bytes_type & bytes);
  » more...
```

Construct an address from a unsigned long in host byte order.

```
explicit address_v4(
    unsigned long addr);
  » more...
```

Copy constructor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
address_v4(
    const address_v4 & other);
  » more...
```

ip::address_v4::address_v4 (1 of 4 overloads)

Default constructor.

```
address_v4();
```

ip::address_v4::address_v4 (2 of 4 overloads)

Construct an address from raw bytes.

```
address_v4(
    const bytes_type & bytes);
```

ip::address_v4::address_v4 (3 of 4 overloads)

Construct an address from a unsigned long in host byte order.

```
address_v4(
    unsigned long addr);
```

ip::address_v4::address_v4 (4 of 4 overloads)

Copy constructor.

```
address_v4(
    const address_v4 & other);
```

ip::address_v4::any

Obtain an address object that represents any address.

```
static address_v4 any();
```

ip::address_v4::broadcast

Obtain an address object that represents the broadcast address.

```
static address_v4 broadcast();
  » more...
```

Obtain an address object that represents the broadcast address that corresponds to the specified address and netmask.

```
static address_v4 broadcast(
    const address_v4 & addr,
    const address_v4 & mask);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::address_v4::broadcast (1 of 2 overloads)

Obtain an address object that represents the broadcast address.

```
static address_v4 broadcast();
```

ip::address_v4::broadcast (2 of 2 overloads)

Obtain an address object that represents the broadcast address that corresponds to the specified address and netmask.

```
static address_v4 broadcast(
    const address_v4 & addr,
    const address_v4 & mask);
```

ip::address_v4::bytes_type

The type used to represent an address as an array of bytes.

```
typedef array< unsigned char, 4 > bytes_type;
```

Remarks

This type is defined in terms of the C++0x template `std::array` when it is available. Otherwise, it uses `boost:array`.

Requirements

Header: `boost/asio/ip/address_v4.hpp`

Convenience header: `boost/asio.hpp`

ip::address_v4::from_string

Create an address from an IP address string in dotted decimal form.

```
static address_v4 from_string(
    const char * str);
  » more...

static address_v4 from_string(
    const char * str,
    boost::system::error_code & ec);
  » more...

static address_v4 from_string(
    const std::string & str);
  » more...

static address_v4 from_string(
    const std::string & str,
    boost::system::error_code & ec);
  » more...
```

ip::address_v4::from_string (1 of 4 overloads)

Create an address from an IP address string in dotted decimal form.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
static address_v4 from_string(
    const char * str);
```

ip::address_v4::from_string (2 of 4 overloads)

Create an address from an IP address string in dotted decimal form.

```
static address_v4 from_string(
    const char * str,
    boost::system::error_code & ec);
```

ip::address_v4::from_string (3 of 4 overloads)

Create an address from an IP address string in dotted decimal form.

```
static address_v4 from_string(
    const std::string & str);
```

ip::address_v4::from_string (4 of 4 overloads)

Create an address from an IP address string in dotted decimal form.

```
static address_v4 from_string(
    const std::string & str,
    boost::system::error_code & ec);
```

ip::address_v4::is_class_a

Determine whether the address is a class A address.

```
bool is_class_a() const;
```

ip::address_v4::is_class_b

Determine whether the address is a class B address.

```
bool is_class_b() const;
```

ip::address_v4::is_class_c

Determine whether the address is a class C address.

```
bool is_class_c() const;
```

ip::address_v4::is_loopback

Determine whether the address is a loopback address.

```
bool is_loopback() const;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::address_v4::is_multicast

Determine whether the address is a multicast address.

```
bool is_multicast() const;
```

ip::address_v4::is_unspecified

Determine whether the address is unspecified.

```
bool is_unspecified() const;
```

ip::address_v4::loopback

Obtain an address object that represents the loopback address.

```
static address_v4 loopback();
```

ip::address_v4::netmask

Obtain the netmask that corresponds to the address, based on its address class.

```
static address_v4 netmask(
    const address_v4 & addr);
```

ip::address_v4::operator!=

Compare two addresses for inequality.

```
friend bool operator!=(
    const address_v4 & a1,
    const address_v4 & a2);
```

Requirements

Header: `boost/asio/ip/address_v4.hpp`

Convenience header: `boost/asio.hpp`

ip::address_v4::operator<

Compare addresses for ordering.

```
friend bool operator<(
    const address_v4 & a1,
    const address_v4 & a2);
```

Requirements

Header: `boost/asio/ip/address_v4.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::address_v4::operator<<

Output an address as a string.

```
template<
    typename Elem,
    typename Traits>
std::basic_ostream< Elem, Traits > & operator<<(
    std::basic_ostream< Elem, Traits > & os,
    const address_v4 & addr);
```

Used to output a human-readable string for a specified address.

Parameters

os The output stream to which the string will be written.

addr The address to be written.

Return Value

The output stream.

ip::address_v4::operator<=

Compare addresses for ordering.

```
friend bool operator<=(
    const address_v4 & a1,
    const address_v4 & a2);
```

Requirements

Header: `boost/asio/ip/address_v4.hpp`

Convenience header: `boost/asio.hpp`

ip::address_v4::operator=

Assign from another address.

```
address_v4 & operator=(
    const address_v4 & other);
```

ip::address_v4::operator==

Compare two addresses for equality.

```
friend bool operator==(
    const address_v4 & a1,
    const address_v4 & a2);
```

Requirements

Header: `boost/asio/ip/address_v4.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::address_v4::operator>

Compare addresses for ordering.

```
friend bool operator>(
    const address_v4 & a1,
    const address_v4 & a2);
```

Requirements

Header: `boost/asio/ip/address_v4.hpp`

Convenience header: `boost/asio.hpp`

ip::address_v4::operator>=

Compare addresses for ordering.

```
friend bool operator>=(
    const address_v4 & a1,
    const address_v4 & a2);
```

Requirements

Header: `boost/asio/ip/address_v4.hpp`

Convenience header: `boost/asio.hpp`

ip::address_v4::to_bytes

Get the address in bytes, in network byte order.

```
bytes_type to_bytes() const;
```

ip::address_v4::to_string

Get the address as a string in dotted decimal format.

```
std::string to_string() const;
  » more...

std::string to_string(
    boost::system::error_code & ec) const;
  » more...
```

ip::address_v4::to_string (1 of 2 overloads)

Get the address as a string in dotted decimal format.

```
std::string to_string() const;
```

ip::address_v4::to_string (2 of 2 overloads)

Get the address as a string in dotted decimal format.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::string to_string(
    boost::system::error_code & ec) const;
```

ip::address_v4::to_ulong

Get the address as an unsigned long in host byte order.

```
unsigned long to_ulong() const;
```

ip::address_v6

Implements IP version 6 style addresses.

```
class address_v6
```

Types

Name	Description
bytes_type	The type used to represent an address as an array of bytes.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
address_v6	Default constructor. Construct an address from raw bytes and scope ID. Copy constructor.
any	Obtain an address object that represents any address.
from_string	Create an address from an IP address string.
is_link_local	Determine whether the address is link local.
is_loopback	Determine whether the address is a loopback address.
is_multicast	Determine whether the address is a multicast address.
is_multicast_global	Determine whether the address is a global multicast address.
is_multicast_link_local	Determine whether the address is a link-local multicast address.
is_multicast_node_local	Determine whether the address is a node-local multicast address.
is_multicast_org_local	Determine whether the address is a org-local multicast address.
is_multicast_site_local	Determine whether the address is a site-local multicast address.
is_site_local	Determine whether the address is site local.
is_unspecified	Determine whether the address is unspecified.
is_v4_compatible	Determine whether the address is an IPv4-compatible address.
is_v4_mapped	Determine whether the address is a mapped IPv4 address.
loopback	Obtain an address object that represents the loopback address.
operator=	Assign from another address.
scope_id	The scope ID of the address.
to_bytes	Get the address in bytes, in network byte order.
to_string	Get the address as a string.
to_v4	Converts an IPv4-mapped or IPv4-compatible address to an IPv4 address.
v4_compatible	Create an IPv4-compatible IPv6 address.
v4_mapped	Create an IPv4-mapped IPv6 address.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Friends

Name	Description
operator!=	Compare two addresses for inequality.
operator<	Compare addresses for ordering.
operator<=	Compare addresses for ordering.
operator==	Compare two addresses for equality.
operator>	Compare addresses for ordering.
operator>=	Compare addresses for ordering.

Related Functions

Name	Description
operator<<	Output an address as a string.

The `ip::address_v6` class provides the ability to use and manipulate IP version 6 addresses.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/address_v6.hpp`

Convenience header: `boost/asio.hpp`

ip::address_v6::address_v6

Default constructor.

```
address_v6();
  » more...
```

Construct an address from raw bytes and scope ID.

```
explicit address_v6(
    const bytes_type & bytes,
    unsigned long scope_id = 0);
  » more...
```

Copy constructor.

```
address_v6(
    const address_v6 & other);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::address_v6::address_v6 (1 of 3 overloads)

Default constructor.

```
address_v6();
```

ip::address_v6::address_v6 (2 of 3 overloads)

Construct an address from raw bytes and scope ID.

```
address_v6(
    const bytes_type & bytes,
    unsigned long scope_id = 0);
```

ip::address_v6::address_v6 (3 of 3 overloads)

Copy constructor.

```
address_v6(
    const address_v6 & other);
```

ip::address_v6::any

Obtain an address object that represents any address.

```
static address_v6 any();
```

ip::address_v6::bytes_type

The type used to represent an address as an array of bytes.

```
typedef array< unsigned char, 16 > bytes_type;
```

Remarks

This type is defined in terms of the C++0x template `std::array` when it is available. Otherwise, it uses `boost:array`.

Requirements

Header: `boost/asio/ip/address_v6.hpp`

Convenience header: `boost/asio.hpp`

ip::address_v6::from_string

Create an address from an IP address string.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
static address_v6 from_string(
    const char * str);
  » more...

static address_v6 from_string(
    const char * str,
    boost::system::error_code & ec);
  » more...

static address_v6 from_string(
    const std::string & str);
  » more...

static address_v6 from_string(
    const std::string & str,
    boost::system::error_code & ec);
  » more...
```

ip::address_v6::from_string (1 of 4 overloads)

Create an address from an IP address string.

```
static address_v6 from_string(
    const char * str);
```

ip::address_v6::from_string (2 of 4 overloads)

Create an address from an IP address string.

```
static address_v6 from_string(
    const char * str,
    boost::system::error_code & ec);
```

ip::address_v6::from_string (3 of 4 overloads)

Create an address from an IP address string.

```
static address_v6 from_string(
    const std::string & str);
```

ip::address_v6::from_string (4 of 4 overloads)

Create an address from an IP address string.

```
static address_v6 from_string(
    const std::string & str,
    boost::system::error_code & ec);
```

ip::address_v6::is_link_local

Determine whether the address is link local.

```
bool is_link_local() const;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::address_v6::is_loopback

Determine whether the address is a loopback address.

```
bool is_loopback() const;
```

ip::address_v6::is_multicast

Determine whether the address is a multicast address.

```
bool is_multicast() const;
```

ip::address_v6::is_multicast_global

Determine whether the address is a global multicast address.

```
bool is_multicast_global() const;
```

ip::address_v6::is_multicast_link_local

Determine whether the address is a link-local multicast address.

```
bool is_multicast_link_local() const;
```

ip::address_v6::is_multicast_node_local

Determine whether the address is a node-local multicast address.

```
bool is_multicast_node_local() const;
```

ip::address_v6::is_multicast_org_local

Determine whether the address is a org-local multicast address.

```
bool is_multicast_org_local() const;
```

ip::address_v6::is_multicast_site_local

Determine whether the address is a site-local multicast address.

```
bool is_multicast_site_local() const;
```

ip::address_v6::is_site_local

Determine whether the address is site local.

```
bool is_site_local() const;
```

ip::address_v6::is_unspecified

Determine whether the address is unspecified.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
bool is_unspecified() const;
```

ip::address_v6::is_v4_compatible

Determine whether the address is an IPv4-compatible address.

```
bool is_v4_compatible() const;
```

ip::address_v6::is_v4_mapped

Determine whether the address is a mapped IPv4 address.

```
bool is_v4_mapped() const;
```

ip::address_v6::loopback

Obtain an address object that represents the loopback address.

```
static address_v6 loopback();
```

ip::address_v6::operator!=

Compare two addresses for inequality.

```
friend bool operator!=(
    const address_v6 & a1,
    const address_v6 & a2);
```

Requirements

Header: `boost/asio/ip/address_v6.hpp`

Convenience header: `boost/asio.hpp`

ip::address_v6::operator<

Compare addresses for ordering.

```
friend bool operator<(
    const address_v6 & a1,
    const address_v6 & a2);
```

Requirements

Header: `boost/asio/ip/address_v6.hpp`

Convenience header: `boost/asio.hpp`

ip::address_v6::operator<<

Output an address as a string.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename Elem,
    typename Traits>
std::basic_ostream< Elem, Traits > & operator<<(
    std::basic_ostream< Elem, Traits > & os,
    const address_v6 & addr);
```

Used to output a human-readable string for a specified address.

Parameters

os The output stream to which the string will be written.

addr The address to be written.

Return Value

The output stream.

ip::address_v6::operator<=

Compare addresses for ordering.

```
friend bool operator<=(
    const address_v6 & a1,
    const address_v6 & a2);
```

Requirements

Header: `boost/asio/ip/address_v6.hpp`

Convenience header: `boost/asio.hpp`

ip::address_v6::operator=

Assign from another address.

```
address_v6 & operator=(
    const address_v6 & other);
```

ip::address_v6::operator==

Compare two addresses for equality.

```
friend bool operator==(
    const address_v6 & a1,
    const address_v6 & a2);
```

Requirements

Header: `boost/asio/ip/address_v6.hpp`

Convenience header: `boost/asio.hpp`

ip::address_v6::operator>

Compare addresses for ordering.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
friend bool operator>(
    const address_v6 & a1,
    const address_v6 & a2);
```

Requirements

Header: `boost/asio/ip/address_v6.hpp`

Convenience header: `boost/asio.hpp`

ip::address_v6::operator>=

Compare addresses for ordering.

```
friend bool operator>=(
    const address_v6 & a1,
    const address_v6 & a2);
```

Requirements

Header: `boost/asio/ip/address_v6.hpp`

Convenience header: `boost/asio.hpp`

ip::address_v6::scope_id

The scope ID of the address.

```
unsigned long scope_id() const;
  » more...

void scope_id(
    unsigned long id);
  » more...
```

ip::address_v6::scope_id (1 of 2 overloads)

The scope ID of the address.

```
unsigned long scope_id() const;
```

Returns the scope ID associated with the IPv6 address.

ip::address_v6::scope_id (2 of 2 overloads)

The scope ID of the address.

```
void scope_id(
    unsigned long id);
```

Modifies the scope ID associated with the IPv6 address.

ip::address_v6::to_bytes

Get the address in bytes, in network byte order.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
bytes_type to_bytes() const;
```

ip::address_v6::to_string

Get the address as a string.

```
std::string to_string() const;
  » more...

std::string to_string(
    boost::system::error_code & ec) const;
  » more...
```

ip::address_v6::to_string (1 of 2 overloads)

Get the address as a string.

```
std::string to_string() const;
```

ip::address_v6::to_string (2 of 2 overloads)

Get the address as a string.

```
std::string to_string(
    boost::system::error_code & ec) const;
```

ip::address_v6::to_v4

Converts an IPv4-mapped or IPv4-compatible address to an IPv4 address.

```
address_v4 to_v4() const;
```

ip::address_v6::v4_compatible

Create an IPv4-compatible IPv6 address.

```
static address_v6 v4_compatible(
    const address_v4 & addr);
```

ip::address_v6::v4_mapped

Create an IPv4-mapped IPv6 address.

```
static address_v6 v4_mapped(
    const address_v4 & addr);
```

ip::basic_endpoint

Describes an endpoint for a version-independent IP socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename InternetProtocol>
class basic_endpoint
```

Types

Name	Description
data_type	The type of the endpoint structure. This type is dependent on the underlying implementation of the socket layer.
protocol_type	The protocol type associated with the endpoint.

Member Functions

Name	Description
address	Get the IP address associated with the endpoint. Set the IP address associated with the endpoint.
basic_endpoint	Default constructor. Construct an endpoint using a port number, specified in the host's byte order. The IP address will be the any address (i.e. INADDR_ANY or in6addr_any). This constructor would typically be used for accepting new connections. Construct an endpoint using a port number and an IP address. This constructor may be used for accepting connections on a specific interface or for making a connection to a remote endpoint. Copy constructor.
capacity	Get the capacity of the endpoint in the native type.
data	Get the underlying endpoint in the native type.
operator=	Assign from another endpoint.
port	Get the port associated with the endpoint. The port number is always in the host's byte order. Set the port associated with the endpoint. The port number is always in the host's byte order.
protocol	The protocol associated with the endpoint.
resize	Set the underlying size of the endpoint in the native type.
size	Get the underlying size of the endpoint in the native type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Friends

Name	Description
operator!=	Compare two endpoints for inequality.
operator<	Compare endpoints for ordering.
operator<=	Compare endpoints for ordering.
operator==	Compare two endpoints for equality.
operator>	Compare endpoints for ordering.
operator>=	Compare endpoints for ordering.

Related Functions

Name	Description
operator<<	Output an endpoint as a string.

The `ip::basic_endpoint` class template describes an endpoint that may be associated with a particular socket.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_endpoint::address

Get the IP address associated with the endpoint.

```
boost::asio::ip::address address() const;
  » more...
```

Set the IP address associated with the endpoint.

```
void address(
    const boost::asio::ip::address & addr);
  » more...
```

ip::basic_endpoint::address (1 of 2 overloads)

Get the IP address associated with the endpoint.

```
boost::asio::ip::address address() const;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::basic_endpoint::address (2 of 2 overloads)

Set the IP address associated with the endpoint.

```
void address(
    const boost::asio::ip::address & addr);
```

ip::basic_endpoint::basic_endpoint

Default constructor.

```
basic_endpoint();
  » more...
```

Construct an endpoint using a port number, specified in the host's byte order. The IP address will be the any address (i.e. INADDR_ANY or in6addr_any). This constructor would typically be used for accepting new connections.

```
basic_endpoint(
    const InternetProtocol & internet_protocol,
    unsigned short port_num);
  » more...
```

Construct an endpoint using a port number and an IP address. This constructor may be used for accepting connections on a specific interface or for making a connection to a remote endpoint.

```
basic_endpoint(
    const boost::asio::ip::address & addr,
    unsigned short port_num);
  » more...
```

Copy constructor.

```
basic_endpoint(
    const basic_endpoint & other);
  » more...
```

ip::basic_endpoint::basic_endpoint (1 of 4 overloads)

Default constructor.

```
basic_endpoint();
```

ip::basic_endpoint::basic_endpoint (2 of 4 overloads)

Construct an endpoint using a port number, specified in the host's byte order. The IP address will be the any address (i.e. INADDR_ANY or in6addr_any). This constructor would typically be used for accepting new connections.

```
basic_endpoint(
    const InternetProtocol & internet_protocol,
    unsigned short port_num);
```

Examples

To initialise an IPv4 TCP endpoint for port 1234, use:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::endpoint ep(boost::asio::ip::tcp::v4(), 1234);
```

To specify an IPv6 UDP endpoint for port 9876, use:

```
boost::asio::ip::udp::endpoint ep(boost::asio::ip::udp::v6(), 9876);
```

ip::basic_endpoint::basic_endpoint (3 of 4 overloads)

Construct an endpoint using a port number and an IP address. This constructor may be used for accepting connections on a specific interface or for making a connection to a remote endpoint.

```
basic_endpoint(
    const boost::asio::ip::address & addr,
    unsigned short port_num);
```

ip::basic_endpoint::basic_endpoint (4 of 4 overloads)

Copy constructor.

```
basic_endpoint(
    const basic_endpoint & other);
```

ip::basic_endpoint::capacity

Get the capacity of the endpoint in the native type.

```
std::size_t capacity() const;
```

ip::basic_endpoint::data

Get the underlying endpoint in the native type.

```
data_type * data();
  » more...

const data_type * data() const;
  » more...
```

ip::basic_endpoint::data (1 of 2 overloads)

Get the underlying endpoint in the native type.

```
data_type * data();
```

ip::basic_endpoint::data (2 of 2 overloads)

Get the underlying endpoint in the native type.

```
const data_type * data() const;
```

ip::basic_endpoint::data_type

The type of the endpoint structure. This type is dependent on the underlying implementation of the socket layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef implementation_defined data_type;
```

Requirements

Header: `boost/asio/ip/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_endpoint::operator!=

Compare two endpoints for inequality.

```
friend bool operator!=(
    const basic_endpoint< InternetProtocol > & e1,
    const basic_endpoint< InternetProtocol > & e2);
```

Requirements

Header: `boost/asio/ip/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_endpoint::operator<

Compare endpoints for ordering.

```
friend bool operator<(
    const basic_endpoint< InternetProtocol > & e1,
    const basic_endpoint< InternetProtocol > & e2);
```

Requirements

Header: `boost/asio/ip/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_endpoint::operator<<

Output an endpoint as a string.

```
std::basic_ostream< Elem, Traits > & operator<<(
    std::basic_ostream< Elem, Traits > & os,
    const basic_endpoint< InternetProtocol > & endpoint);
```

Used to output a human-readable string for a specified endpoint.

Parameters

os	The output stream to which the string will be written.
endpoint	The endpoint to be written.

Return Value

The output stream.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::basic_endpoint::operator<=

Compare endpoints for ordering.

```
friend bool operator<=(
    const basic_endpoint< InternetProtocol > & e1,
    const basic_endpoint< InternetProtocol > & e2);
```

Requirements

Header: `boost/asio/ip/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_endpoint::operator=

Assign from another endpoint.

```
basic_endpoint & operator=(
    const basic_endpoint & other);
```

ip::basic_endpoint::operator==

Compare two endpoints for equality.

```
friend bool operator==(
    const basic_endpoint< InternetProtocol > & e1,
    const basic_endpoint< InternetProtocol > & e2);
```

Requirements

Header: `boost/asio/ip/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_endpoint::operator>

Compare endpoints for ordering.

```
friend bool operator>(
    const basic_endpoint< InternetProtocol > & e1,
    const basic_endpoint< InternetProtocol > & e2);
```

Requirements

Header: `boost/asio/ip/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_endpoint::operator>=

Compare endpoints for ordering.

```
friend bool operator>=(
    const basic_endpoint< InternetProtocol > & e1,
    const basic_endpoint< InternetProtocol > & e2);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/ip/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_endpoint::port

Get the port associated with the endpoint. The port number is always in the host's byte order.

```
unsigned short port() const;
  » more...
```

Set the port associated with the endpoint. The port number is always in the host's byte order.

```
void port(
    unsigned short port_num);
  » more...
```

ip::basic_endpoint::port (1 of 2 overloads)

Get the port associated with the endpoint. The port number is always in the host's byte order.

```
unsigned short port() const;
```

ip::basic_endpoint::port (2 of 2 overloads)

Set the port associated with the endpoint. The port number is always in the host's byte order.

```
void port(
    unsigned short port_num);
```

ip::basic_endpoint::protocol

The protocol associated with the endpoint.

```
protocol_type protocol() const;
```

ip::basic_endpoint::protocol_type

The protocol type associated with the endpoint.

```
typedef InternetProtocol protocol_type;
```

Requirements

Header: `boost/asio/ip/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_endpoint::resize

Set the underlying size of the endpoint in the native type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void resize(
    std::size_t new_size);
```

ip::basic_endpoint::size

Get the underlying size of the endpoint in the native type.

```
std::size_t size() const;
```

ip::basic_resolver

Provides endpoint resolution functionality.

```
template<
    typename InternetProtocol,
    typename ResolverService = resolver_service<InternetProtocol>>
class basic_resolver :
  public basic_io_object< ResolverService >
```

Types

Name	Description
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
iterator	The iterator type.
protocol_type	The protocol type.
query	The query type.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
async_resolve	Asynchronously perform forward resolution of a query to a list of entries. Asynchronously perform reverse resolution of an endpoint to a list of entries.
basic_resolver	Constructor.
cancel	Cancel any asynchronous operations that are waiting on the resolver.
get_io_service	Get the io_service associated with the object.
resolve	Perform forward resolution of a query to a list of entries. Perform reverse resolution of an endpoint to a list of entries.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `ip::basic_resolver` class template provides the ability to resolve a query to a list of endpoints.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/basic_resolver.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver::async_resolve

Asynchronously perform forward resolution of a query to a list of entries.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ResolveHandler>
void async_resolve(
    const query & q,
    ResolveHandler handler);
  » more...
```

Asynchronously perform reverse resolution of an endpoint to a list of entries.

```
template<
    typename ResolveHandler>
void async_resolve(
    const endpoint_type & e,
    ResolveHandler handler);
  » more...
```

ip::basic_resolver::async_resolve (1 of 2 overloads)

Asynchronously perform forward resolution of a query to a list of entries.

```
template<
    typename ResolveHandler>
void async_resolve(
    const query & q,
    ResolveHandler handler);
```

This function is used to asynchronously resolve a query into a list of endpoint entries.

Parameters

q — A query object that determines what endpoints will be returned.

handler — The handler to be called when the resolve operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  resolver::iterator iterator             // Forward-only iterator that can
                                          // be used to traverse the list
                                          // of endpoint entries.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

A default constructed iterator represents the end of the list.

A successful resolve operation is guaranteed to pass at least one entry to the handler.

ip::basic_resolver::async_resolve (2 of 2 overloads)

Asynchronously perform reverse resolution of an endpoint to a list of entries.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ResolveHandler>
void async_resolve(
    const endpoint_type & e,
    ResolveHandler handler);
```

This function is used to asynchronously resolve an endpoint into a list of endpoint entries.

Parameters

e — An endpoint object that determines what endpoints will be returned.

handler — The handler to be called when the resolve operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  resolver::iterator iterator             // Forward-only iterator that can
                                          // be used to traverse the list
                                          // of endpoint entries.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

A default constructed iterator represents the end of the list.

A successful resolve operation is guaranteed to pass at least one entry to the handler.

ip::basic_resolver::basic_resolver

Constructor.

```
basic_resolver(
    boost::asio::io_service & io_service);
```

This constructor creates a `ip::basic_resolver`.

Parameters

io_service — The `io_service` object that the resolver will use to dispatch handlers for any asynchronous operations performed on the timer.

ip::basic_resolver::cancel

Cancel any asynchronous operations that are waiting on the resolver.

```
void cancel();
```

This function forces the completion of any pending asynchronous operations on the host resolver. The handler for each cancelled operation will be invoked with the `boost::asio::error::operation_aborted` error code.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::basic_resolver::endpoint_type

The endpoint type.

```
typedef InternetProtocol::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/ip/basic_resolver.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

ip::basic_resolver::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

ip::basic_resolver::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

ip::basic_resolver::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

ip::basic_resolver::get_service

Get the service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

ip::basic_resolver::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

ip::basic_resolver::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

ip::basic_resolver::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

ip::basic_resolver::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/ip/basic_resolver.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver::iterator

The iterator type.

```
typedef basic_resolver_iterator< InternetProtocol > iterator;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
difference_type	The type used for the distance between two iterators.
iterator_category	The iterator category.
pointer	The type of the result of applying operator->() to the iterator.
reference	The type of the result of applying operator*() to the iterator.
value_type	The type of the value pointed to by the iterator.

Member Functions

Name	Description
basic_resolver_iterator	Default constructor creates an end iterator.
create	Create an iterator from an addrinfo list returned by getaddrinfo. Create an iterator from an endpoint, host name and service name.
operator *	Dereference an iterator.
operator++	Increment operator (prefix). Increment operator (postfix).
operator->	Dereference an iterator.

Friends

Name	Description
operator!=	Test two iterators for inequality.
operator==	Test two iterators for equality.

The `ip::basic_resolver_iterator` class template is used to define iterators over the results returned by a resolver.

The iterator's value_type, obtained when the iterator is dereferenced, is:

```
const basic_resolver_entry<InternetProtocol>
```

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/basic_resolver.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::basic_resolver::protocol_type

The protocol type.

```
typedef InternetProtocol protocol_type;
```

Requirements

Header: `boost/asio/ip/basic_resolver.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver::query

The query type.

```
typedef basic_resolver_query< InternetProtocol > query;
```

Types

Name	Description
flags	A bitmask type (C++ Std [lib.bitmask.types]).
protocol_type	The protocol type associated with the endpoint query.

Member Functions

Name	Description
basic_resolver_query	Construct with specified service name for any protocol. Construct with specified service name for a given protocol. Construct with specified host name and service name for any protocol. Construct with specified host name and service name for a given protocol.
hints	Get the hints associated with the query.
host_name	Get the host name associated with the query.
service_name	Get the service name associated with the query.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Data Members

Name	Description
address_configured	Only return IPv4 addresses if a non-loopback IPv4 address is configured for the system. Only return IPv6 addresses if a non-loopback IPv6 address is configured for the system.
all_matching	If used with v4_mapped, return all matching IPv6 and IPv4 addresses.
canonical_name	Determine the canonical name of the host specified in the query.
numeric_host	Host name should be treated as a numeric string defining an IPv4 or IPv6 address and no name resolution should be attempted.
numeric_service	Service name should be treated as a numeric string defining a port number and no name resolution should be attempted.
passive	Indicate that returned endpoint is intended for use as a locally bound socket endpoint.
v4_mapped	If the query protocol family is specified as IPv6, return IPv4-mapped IPv6 addresses on finding no IPv6 addresses.

The `ip::basic_resolver_query` class template describes a query that can be passed to a resolver.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/basic_resolver.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver::resolve

Perform forward resolution of a query to a list of entries.

```
iterator resolve(
    const query & q);
  » more...

iterator resolve(
    const query & q,
    boost::system::error_code & ec);
  » more...
```

Perform reverse resolution of an endpoint to a list of entries.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
iterator resolve(
    const endpoint_type & e);
  » more...

iterator resolve(
    const endpoint_type & e,
    boost::system::error_code & ec);
  » more...
```

ip::basic_resolver::resolve (1 of 4 overloads)

Perform forward resolution of a query to a list of entries.

```
iterator resolve(
    const query & q);
```

This function is used to resolve a query into a list of endpoint entries.

Parameters

q A query object that determines what endpoints will be returned.

Return Value

A forward-only iterator that can be used to traverse the list of endpoint entries.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

A default constructed iterator represents the end of the list.

A successful call to this function is guaranteed to return at least one entry.

ip::basic_resolver::resolve (2 of 4 overloads)

Perform forward resolution of a query to a list of entries.

```
iterator resolve(
    const query & q,
    boost::system::error_code & ec);
```

This function is used to resolve a query into a list of endpoint entries.

Parameters

q A query object that determines what endpoints will be returned.

ec Set to indicate what error occurred, if any.

Return Value

A forward-only iterator that can be used to traverse the list of endpoint entries. Returns a default constructed iterator if an error occurs.

Remarks

A default constructed iterator represents the end of the list.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

A successful call to this function is guaranteed to return at least one entry.

ip::basic_resolver::resolve (3 of 4 overloads)

Perform reverse resolution of an endpoint to a list of entries.

```
iterator resolve(
    const endpoint_type & e);
```

This function is used to resolve an endpoint into a list of endpoint entries.

Parameters

e An endpoint object that determines what endpoints will be returned.

Return Value

A forward-only iterator that can be used to traverse the list of endpoint entries.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

A default constructed iterator represents the end of the list.

A successful call to this function is guaranteed to return at least one entry.

ip::basic_resolver::resolve (4 of 4 overloads)

Perform reverse resolution of an endpoint to a list of entries.

```
iterator resolve(
    const endpoint_type & e,
    boost::system::error_code & ec);
```

This function is used to resolve an endpoint into a list of endpoint entries.

Parameters

e An endpoint object that determines what endpoints will be returned.

ec Set to indicate what error occurred, if any.

Return Value

A forward-only iterator that can be used to traverse the list of endpoint entries. Returns a default constructed iterator if an error occurs.

Remarks

A default constructed iterator represents the end of the list.

A successful call to this function is guaranteed to return at least one entry.

ip::basic_resolver::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

ip::basic_resolver::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef ResolverService service_type;
```

Requirements

Header: `boost/asio/ip/basic_resolver.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver_entry

An entry produced by a resolver.

```
template<
    typename InternetProtocol>
class basic_resolver_entry
```

Types

Name	Description
endpoint_type	The endpoint type associated with the endpoint entry.
protocol_type	The protocol type associated with the endpoint entry.

Member Functions

Name	Description
basic_resolver_entry	Default constructor. Construct with specified endpoint, host name and service name.
endpoint	Get the endpoint associated with the entry.
host_name	Get the host name associated with the entry.
operator endpoint_type	Convert to the endpoint associated with the entry.
service_name	Get the service name associated with the entry.

The `ip::basic_resolver_entry` class template describes an entry as returned by a resolver.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/basic_resolver_entry.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver_entry::basic_resolver_entry

Default constructor.

```
basic_resolver_entry();
  » more...
```

Construct with specified endpoint, host name and service name.

```
basic_resolver_entry(
    const endpoint_type & ep,
    const std::string & host,
    const std::string & service);
  » more...
```

ip::basic_resolver_entry::basic_resolver_entry (1 of 2 overloads)

Default constructor.

```
basic_resolver_entry();
```

ip::basic_resolver_entry::basic_resolver_entry (2 of 2 overloads)

Construct with specified endpoint, host name and service name.

```
basic_resolver_entry(
    const endpoint_type & ep,
    const std::string & host,
    const std::string & service);
```

ip::basic_resolver_entry::endpoint

Get the endpoint associated with the entry.

```
endpoint_type endpoint() const;
```

ip::basic_resolver_entry::endpoint_type

The endpoint type associated with the endpoint entry.

```
typedef InternetProtocol::endpoint endpoint_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/ip/basic_resolver_entry.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver_entry::host_name

Get the host name associated with the entry.

```
std::string host_name() const;
```

ip::basic_resolver_entry::operator endpoint_type

Convert to the endpoint associated with the entry.

```
operator endpoint_type() const;
```

ip::basic_resolver_entry::protocol_type

The protocol type associated with the endpoint entry.

```
typedef InternetProtocol protocol_type;
```

Requirements

Header: `boost/asio/ip/basic_resolver_entry.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver_entry::service_name

Get the service name associated with the entry.

```
std::string service_name() const;
```

ip::basic_resolver_iterator

An iterator over the entries produced by a resolver.

```
template<
    typename InternetProtocol>
class basic_resolver_iterator
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
difference_type	The type used for the distance between two iterators.
iterator_category	The iterator category.
pointer	The type of the result of applying operator->() to the iterator.
reference	The type of the result of applying operator*() to the iterator.
value_type	The type of the value pointed to by the iterator.

Member Functions

Name	Description
basic_resolver_iterator	Default constructor creates an end iterator.
create	Create an iterator from an addrinfo list returned by getaddrinfo. Create an iterator from an endpoint, host name and service name.
operator *	Dereference an iterator.
operator++	Increment operator (prefix). Increment operator (postfix).
operator->	Dereference an iterator.

Friends

Name	Description
operator!=	Test two iterators for inequality.
operator==	Test two iterators for equality.

The `ip::basic_resolver_iterator` class template is used to define iterators over the results returned by a resolver.

The iterator's value_type, obtained when the iterator is dereferenced, is:

```
const basic_resolver_entry<InternetProtocol>
```

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/basic_resolver_iterator.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio.hpp`

ip::basic_resolver_iterator::basic_resolver_iterator

Default constructor creates an end iterator.

```
basic_resolver_iterator();
```

ip::basic_resolver_iterator::create

Create an iterator from an addrinfo list returned by getaddrinfo.

```
static basic_resolver_iterator create(
    boost::asio::detail::addrinfo_type * address_info,
    const std::string & host_name,
    const std::string & service_name);
  » more...
```

Create an iterator from an endpoint, host name and service name.

```
static basic_resolver_iterator create(
    const typename InternetProtocol::endpoint & endpoint,
    const std::string & host_name,
    const std::string & service_name);
  » more...
```

ip::basic_resolver_iterator::create (1 of 2 overloads)

Create an iterator from an addrinfo list returned by getaddrinfo.

```
static basic_resolver_iterator create(
    boost::asio::detail::addrinfo_type * address_info,
    const std::string & host_name,
    const std::string & service_name);
```

ip::basic_resolver_iterator::create (2 of 2 overloads)

Create an iterator from an endpoint, host name and service name.

```
static basic_resolver_iterator create(
    const typename InternetProtocol::endpoint & endpoint,
    const std::string & host_name,
    const std::string & service_name);
```

ip::basic_resolver_iterator::difference_type

The type used for the distance between two iterators.

```
typedef std::ptrdiff_t difference_type;
```

Requirements

Header: `boost/asio/ip/basic_resolver_iterator.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::basic_resolver_iterator::iterator_category

The iterator category.

```
typedef std::forward_iterator_tag iterator_category;
```

Requirements

Header: `boost/asio/ip/basic_resolver_iterator.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver_iterator::operator *

Dereference an iterator.

```
const basic_resolver_entry< InternetProtocol > & operator *() const;
```

ip::basic_resolver_iterator::operator!=

Test two iterators for inequality.

```
friend bool operator!=(
    const basic_resolver_iterator & a,
    const basic_resolver_iterator & b);
```

Requirements

Header: `boost/asio/ip/basic_resolver_iterator.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver_iterator::operator++

Increment operator (prefix).

```
basic_resolver_iterator & operator++();
  » more...
```

Increment operator (postfix).

```
basic_resolver_iterator operator++(
    int );
  » more...
```

ip::basic_resolver_iterator::operator++ (1 of 2 overloads)

Increment operator (prefix).

```
basic_resolver_iterator & operator++();
```

ip::basic_resolver_iterator::operator++ (2 of 2 overloads)

Increment operator (postfix).

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
basic_resolver_iterator operator++(
    int );
```

ip::basic_resolver_iterator::operator->

Dereference an iterator.

```
const basic_resolver_entry< InternetProtocol > * operator->() const;
```

ip::basic_resolver_iterator::operator==

Test two iterators for equality.

```
friend bool operator==(
    const basic_resolver_iterator & a,
    const basic_resolver_iterator & b);
```

Requirements

Header: `boost/asio/ip/basic_resolver_iterator.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver_iterator::pointer

The type of the result of applying `operator->()` to the iterator.

```
typedef const basic_resolver_entry< InternetProtocol > * pointer;
```

Requirements

Header: `boost/asio/ip/basic_resolver_iterator.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver_iterator::reference

The type of the result of applying `operator*()` to the iterator.

```
typedef const basic_resolver_entry< InternetProtocol > & reference;
```

Types

Name	Description
endpoint_type	The endpoint type associated with the endpoint entry.
protocol_type	The protocol type associated with the endpoint entry.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
basic_resolver_entry	Default constructor. Construct with specified endpoint, host name and service name.
endpoint	Get the endpoint associated with the entry.
host_name	Get the host name associated with the entry.
operator endpoint_type	Convert to the endpoint associated with the entry.
service_name	Get the service name associated with the entry.

The `ip::basic_resolver_entry` class template describes an entry as returned by a resolver.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/basic_resolver_iterator.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver_iterator::value_type

The type of the value pointed to by the iterator.

```
typedef basic_resolver_entry< InternetProtocol > value_type;
```

Types

Name	Description
endpoint_type	The endpoint type associated with the endpoint entry.
protocol_type	The protocol type associated with the endpoint entry.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
basic_resolver_entry	Default constructor. Construct with specified endpoint, host name and service name.
endpoint	Get the endpoint associated with the entry.
host_name	Get the host name associated with the entry.
operator endpoint_type	Convert to the endpoint associated with the entry.
service_name	Get the service name associated with the entry.

The `ip::basic_resolver_entry` class template describes an entry as returned by a resolver.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/basic_resolver_iterator.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver_query

An query to be passed to a resolver.

```
template<
    typename InternetProtocol>
class basic_resolver_query :
  public ip::resolver_query_base
```

Types

Name	Description
flags	A bitmask type (C++ Std [lib.bitmask.types]).
protocol_type	The protocol type associated with the endpoint query.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
basic_resolver_query	Construct with specified service name for any protocol. Construct with specified service name for a given protocol. Construct with specified host name and service name for any protocol. Construct with specified host name and service name for a given protocol.
hints	Get the hints associated with the query.
host_name	Get the host name associated with the query.
service_name	Get the service name associated with the query.

Data Members

Name	Description
address_configured	Only return IPv4 addresses if a non-loopback IPv4 address is configured for the system. Only return IPv6 addresses if a non-loopback IPv6 address is configured for the system.
all_matching	If used with v4_mapped, return all matching IPv6 and IPv4 addresses.
canonical_name	Determine the canonical name of the host specified in the query.
numeric_host	Host name should be treated as a numeric string defining an IPv4 or IPv6 address and no name resolution should be attempted.
numeric_service	Service name should be treated as a numeric string defining a port number and no name resolution should be attempted.
passive	Indicate that returned endpoint is intended for use as a locally bound socket endpoint.
v4_mapped	If the query protocol family is specified as IPv6, return IPv4-mapped IPv6 addresses on finding no IPv6 addresses.

The `ip::basic_resolver_query` class template describes a query that can be passed to a resolver.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/basic_resolver_query.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio.hpp`

ip::basic_resolver_query::address_configured

Inherited from ip::resolver_query_base.

Only return IPv4 addresses if a non-loopback IPv4 address is configured for the system. Only return IPv6 addresses if a non-loopback IPv6 address is configured for the system.

```
static const flags address_configured = implementation_defined;
```

ip::basic_resolver_query::all_matching

Inherited from ip::resolver_query_base.

If used with v4_mapped, return all matching IPv6 and IPv4 addresses.

```
static const flags all_matching = implementation_defined;
```

ip::basic_resolver_query::basic_resolver_query

Construct with specified service name for any protocol.

```
basic_resolver_query(
    const std::string & service,
    resolver_query_base::flags resolve_flags = passive|address_configured);
  » more...
```

Construct with specified service name for a given protocol.

```
basic_resolver_query(
    const protocol_type & protocol,
    const std::string & service,
    resolver_query_base::flags resolve_flags = passive|address_configured);
  » more...
```

Construct with specified host name and service name for any protocol.

```
basic_resolver_query(
    const std::string & host,
    const std::string & service,
    resolver_query_base::flags resolve_flags = address_configured);
  » more...
```

Construct with specified host name and service name for a given protocol.

```
basic_resolver_query(
    const protocol_type & protocol,
    const std::string & host,
    const std::string & service,
    resolver_query_base::flags resolve_flags = address_configured);
  » more...
```

ip::basic_resolver_query::basic_resolver_query (1 of 4 overloads)

Construct with specified service name for any protocol.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
basic_resolver_query(
    const std::string & service,
    resolver_query_base::flags resolve_flags = passive|address_configured);
```

This constructor is typically used to perform name resolution for local service binding.

Parameters

service	A string identifying the requested service. This may be a descriptive name or a numeric string corresponding to a port number.
resolve_flags	A set of flags that determine how name resolution should be performed. The default flags are suitable for local service binding.

Remarks

On POSIX systems, service names are typically defined in the file `/etc/services`. On Windows, service names may be found in the file `c:\windows\system32\drivers\etc\services`. Operating systems may use additional locations when resolving service names.

ip::basic_resolver_query::basic_resolver_query (2 of 4 overloads)

Construct with specified service name for a given protocol.

```
basic_resolver_query(
    const protocol_type & protocol,
    const std::string & service,
    resolver_query_base::flags resolve_flags = passive|address_configured);
```

This constructor is typically used to perform name resolution for local service binding with a specific protocol version.

Parameters

protocol	A protocol object, normally representing either the IPv4 or IPv6 version of an internet protocol.
service	A string identifying the requested service. This may be a descriptive name or a numeric string corresponding to a port number.
resolve_flags	A set of flags that determine how name resolution should be performed. The default flags are suitable for local service binding.

Remarks

On POSIX systems, service names are typically defined in the file `/etc/services`. On Windows, service names may be found in the file `c:\windows\system32\drivers\etc\services`. Operating systems may use additional locations when resolving service names.

ip::basic_resolver_query::basic_resolver_query (3 of 4 overloads)

Construct with specified host name and service name for any protocol.

```
basic_resolver_query(
    const std::string & host,
    const std::string & service,
    resolver_query_base::flags resolve_flags = address_configured);
```

This constructor is typically used to perform name resolution for communication with remote hosts.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

host	A string identifying a location. May be a descriptive name or a numeric address string. If an empty string and the passive flag has been specified, the resolved endpoints are suitable for local service binding. If an empty string and passive is not specified, the resolved endpoints will use the loopback address.
service	A string identifying the requested service. This may be a descriptive name or a numeric string corresponding to a port number. May be an empty string, in which case all resolved endpoints will have a port number of 0.
resolve_flags	A set of flags that determine how name resolution should be performed. The default flags are suitable for communication with remote hosts.

Remarks

On POSIX systems, host names may be locally defined in the file `/etc/hosts`. On Windows, host names may be defined in the file `c:\windows\system32\drivers\etc\hosts`. Remote host name resolution is performed using DNS. Operating systems may use additional locations when resolving host names (such as NETBIOS names on Windows).

On POSIX systems, service names are typically defined in the file `/etc/services`. On Windows, service names may be found in the file `c:\windows\system32\drivers\etc\services`. Operating systems may use additional locations when resolving service names.

ip::basic_resolver_query::basic_resolver_query (4 of 4 overloads)

Construct with specified host name and service name for a given protocol.

```
basic_resolver_query(
    const protocol_type & protocol,
    const std::string & host,
    const std::string & service,
    resolver_query_base::flags resolve_flags = address_configured);
```

This constructor is typically used to perform name resolution for communication with remote hosts.

Parameters

protocol	A protocol object, normally representing either the IPv4 or IPv6 version of an internet protocol.
host	A string identifying a location. May be a descriptive name or a numeric address string. If an empty string and the passive flag has been specified, the resolved endpoints are suitable for local service binding. If an empty string and passive is not specified, the resolved endpoints will use the loopback address.
service	A string identifying the requested service. This may be a descriptive name or a numeric string corresponding to a port number. May be an empty string, in which case all resolved endpoints will have a port number of 0.
resolve_flags	A set of flags that determine how name resolution should be performed. The default flags are suitable for communication with remote hosts.

Remarks

On POSIX systems, host names may be locally defined in the file `/etc/hosts`. On Windows, host names may be defined in the file `c:\windows\system32\drivers\etc\hosts`. Remote host name resolution is performed using DNS. Operating systems may use additional locations when resolving host names (such as NETBIOS names on Windows).

On POSIX systems, service names are typically defined in the file `/etc/services`. On Windows, service names may be found in the file `c:\windows\system32\drivers\etc\services`. Operating systems may use additional locations when resolving service names.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::basic_resolver_query::canonical_name

Inherited from ip::resolver_query_base.

Determine the canonical name of the host specified in the query.

```
static const flags canonical_name = implementation_defined;
```

ip::basic_resolver_query::flags

Inherited from ip::resolver_query_base.

A bitmask type (C++ Std [lib.bitmask.types]).

```
typedef unspecified flags;
```

Requirements

Header: `boost/asio/ip/basic_resolver_query.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver_query::hints

Get the hints associated with the query.

```
const boost::asio::detail::addrinfo_type & hints() const;
```

ip::basic_resolver_query::host_name

Get the host name associated with the query.

```
std::string host_name() const;
```

ip::basic_resolver_query::numeric_host

Inherited from ip::resolver_query_base.

Host name should be treated as a numeric string defining an IPv4 or IPv6 address and no name resolution should be attempted.

```
static const flags numeric_host = implementation_defined;
```

ip::basic_resolver_query::numeric_service

Inherited from ip::resolver_query_base.

Service name should be treated as a numeric string defining a port number and no name resolution should be attempted.

```
static const flags numeric_service = implementation_defined;
```

ip::basic_resolver_query::passive

Inherited from ip::resolver_query_base.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Indicate that returned endpoint is intended for use as a locally bound socket endpoint.

```
static const flags passive = implementation_defined;
```

ip::basic_resolver_query::protocol_type

The protocol type associated with the endpoint query.

```
typedef InternetProtocol protocol_type;
```

Requirements

Header: `boost/asio/ip/basic_resolver_query.hpp`

Convenience header: `boost/asio.hpp`

ip::basic_resolver_query::service_name

Get the service name associated with the query.

```
std::string service_name() const;
```

ip::basic_resolver_query::v4_mapped

Inherited from ip::resolver_query_base.

If the query protocol family is specified as IPv6, return IPv4-mapped IPv6 addresses on finding no IPv6 addresses.

```
static const flags v4_mapped = implementation_defined;
```

ip::host_name

Get the current host name.

```
std::string host_name();
  » more...

std::string host_name(
    boost::system::error_code & ec);
  » more...
```

Requirements

Header: `boost/asio/ip/host_name.hpp`

Convenience header: `boost/asio.hpp`

ip::host_name (1 of 2 overloads)

Get the current host name.

```
std::string host_name();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::host_name (2 of 2 overloads)

Get the current host name.

```
std::string host_name(
    boost::system::error_code & ec);
```

ip::icmp

Encapsulates the flags needed for ICMP.

```
class icmp
```

Types

Name	Description
endpoint	The type of a ICMP endpoint.
resolver	The ICMP resolver type.
socket	The ICMP socket type.

Member Functions

Name	Description
family	Obtain an identifier for the protocol family.
protocol	Obtain an identifier for the protocol.
type	Obtain an identifier for the type of the protocol.
v4	Construct to represent the IPv4 ICMP protocol.
v6	Construct to represent the IPv6 ICMP protocol.

Friends

Name	Description
operator!=	Compare two protocols for inequality.
operator==	Compare two protocols for equality.

The `ip::icmp` class contains flags necessary for ICMP sockets.

Thread Safety

Distinct objects: Safe.

Shared objects: Safe.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/ip/icmp.hpp`

Convenience header: `boost/asio.hpp`

ip::icmp::endpoint

The type of a ICMP endpoint.

```
typedef basic_endpoint< icmp > endpoint;
```

Types

Name	Description
data_type	The type of the endpoint structure. This type is dependent on the underlying implementation of the socket layer.
protocol_type	The protocol type associated with the endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
address	Get the IP address associated with the endpoint. Set the IP address associated with the endpoint.
basic_endpoint	Default constructor. Construct an endpoint using a port number, specified in the host's byte order. The IP address will be the any address (i.e. INADDR_ANY or in6addr_any). This constructor would typically be used for accepting new connections. Construct an endpoint using a port number and an IP address. This constructor may be used for accepting connections on a specific interface or for making a connection to a remote endpoint. Copy constructor.
capacity	Get the capacity of the endpoint in the native type.
data	Get the underlying endpoint in the native type.
operator=	Assign from another endpoint.
port	Get the port associated with the endpoint. The port number is always in the host's byte order. Set the port associated with the endpoint. The port number is always in the host's byte order.
protocol	The protocol associated with the endpoint.
resize	Set the underlying size of the endpoint in the native type.
size	Get the underlying size of the endpoint in the native type.

Friends

Name	Description
operator!=	Compare two endpoints for inequality.
operator<	Compare endpoints for ordering.
operator<=	Compare endpoints for ordering.
operator==	Compare two endpoints for equality.
operator>	Compare endpoints for ordering.
operator>=	Compare endpoints for ordering.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Related Functions

Name	Description
operator<<	Output an endpoint as a string.

The `ip::basic_endpoint` class template describes an endpoint that may be associated with a particular socket.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/icmp.hpp`

Convenience header: `boost/asio.hpp`

ip::icmp::family

Obtain an identifier for the protocol family.

```
int family() const;
```

ip::icmp::operator!=

Compare two protocols for inequality.

```
friend bool operator!=(
    const icmp & p1,
    const icmp & p2);
```

Requirements

Header: `boost/asio/ip/icmp.hpp`

Convenience header: `boost/asio.hpp`

ip::icmp::operator==

Compare two protocols for equality.

```
friend bool operator==(
    const icmp & p1,
    const icmp & p2);
```

Requirements

Header: `boost/asio/ip/icmp.hpp`

Convenience header: `boost/asio.hpp`

ip::icmp::protocol

Obtain an identifier for the protocol.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
int protocol() const;
```

ip::icmp::resolver

The ICMP resolver type.

```
typedef basic_resolver< icmp > resolver;
```

Types

Name	Description
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
iterator	The iterator type.
protocol_type	The protocol type.
query	The query type.
service_type	The type of the service that will be used to provide I/O operations.

Member Functions

Name	Description
async_resolve	Asynchronously perform forward resolution of a query to a list of entries. Asynchronously perform reverse resolution of an endpoint to a list of entries.
basic_resolver	Constructor.
cancel	Cancel any asynchronous operations that are waiting on the resolver.
get_io_service	Get the io_service associated with the object.
resolve	Perform forward resolution of a query to a list of entries. Perform reverse resolution of an endpoint to a list of entries.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `ip::basic_resolver` class template provides the ability to resolve a query to a list of endpoints.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/icmp.hpp`

Convenience header: `boost/asio.hpp`

ip::icmp::socket

The ICMP socket type.

```
typedef basic_raw_socket< icmp > socket;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
async_receive	Start an asynchronous receive on a connected socket.
async_receive_from	Start an asynchronous receive.
async_send	Start an asynchronous send on a connected socket.
async_send_to	Start an asynchronous send.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_raw_socket	Construct a basic_raw_socket without opening it. Construct and open a basic_raw_socket. Construct a basic_raw_socket, opening it and binding it to the given local endpoint. Construct a basic_raw_socket on an existing native socket. Move-construct a basic_raw_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_raw_socket from another.
receive	Receive some data on a connected socket.
receive_from	Receive raw data with the endpoint of the sender.
remote_endpoint	Get the remote endpoint of the socket.
send	Send some data on a connected socket.
send_to	Send raw data to the specified endpoint.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_raw_socket` class template provides asynchronous and blocking raw-oriented socket functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/icmp.hpp`

Convenience header: `boost/asio.hpp`

ip::icmp::type

Obtain an identifier for the type of the protocol.

```
int type() const;
```

ip::icmp::v4

Construct to represent the IPv4 ICMP protocol.

```
static icmp v4();
```

ip::icmp::v6

Construct to represent the IPv6 ICMP protocol.

```
static icmp v6();
```

ip::multicast::enable_loopback

Socket option determining whether outgoing multicast packets will be received on the same socket if it is a member of the multicast group.

```
typedef implementation_defined enable_loopback;
```

Implements the IPPROTO_IP/IP_MULTICAST_LOOP socket option.

Examples

Setting the option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::ip::multicast::enable_loopback option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::ip::multicast::enable_loopback option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/ip/multicast.hpp`

Convenience header: `boost/asio.hpp`

ip::multicast::hops

Socket option for time-to-live associated with outgoing multicast packets.

```
typedef implementation_defined hops;
```

Implements the IPPROTO_IP/IP_MULTICAST_TTL socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::ip::multicast::hops option(4);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::ip::multicast::hops option;
socket.get_option(option);
int ttl = option.value();
```

Requirements

Header: `boost/asio/ip/multicast.hpp`

Convenience header: `boost/asio.hpp`

ip::multicast::join_group

Socket option to join a multicast group on a specified interface.

```
typedef implementation_defined join_group;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Implements the IPPROTO_IP/IP_ADD_MEMBERSHIP socket option.

Examples

Setting the option to join a multicast group:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::ip::address multicast_address =
  boost::asio::ip::address::from_string("225.0.0.1");
boost::asio::ip::multicast::join_group option(multicast_address);
socket.set_option(option);
```

Requirements

Header: `boost/asio/ip/multicast.hpp`

Convenience header: `boost/asio.hpp`

ip::multicast::leave_group

Socket option to leave a multicast group on a specified interface.

```
typedef implementation_defined leave_group;
```

Implements the IPPROTO_IP/IP_DROP_MEMBERSHIP socket option.

Examples

Setting the option to leave a multicast group:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::ip::address multicast_address =
  boost::asio::ip::address::from_string("225.0.0.1");
boost::asio::ip::multicast::leave_group option(multicast_address);
socket.set_option(option);
```

Requirements

Header: `boost/asio/ip/multicast.hpp`

Convenience header: `boost/asio.hpp`

ip::multicast::outbound_interface

Socket option for local interface to use for outgoing multicast packets.

```
typedef implementation_defined outbound_interface;
```

Implements the IPPROTO_IP/IP_MULTICAST_IF socket option.

Examples

Setting the option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::ip::address_v4 local_interface =
  boost::asio::ip::address_v4::from_string("1.2.3.4");
boost::asio::ip::multicast::outbound_interface option(local_interface);
socket.set_option(option);
```

Requirements

Header: `boost/asio/ip/multicast.hpp`

Convenience header: `boost/asio.hpp`

ip::resolver_query_base

The `ip::resolver_query_base` class is used as a base for the `ip::basic_resolver_query` class templates to provide a common place to define the flag constants.

```
class resolver_query_base
```

Types

Name	Description
flags	A bitmask type (C++ Std [lib.bitmask.types]).

Protected Member Functions

Name	Description
~resolver_query_base	Protected destructor to prevent deletion through this type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Data Members

Name	Description
address_configured	Only return IPv4 addresses if a non-loopback IPv4 address is configured for the system. Only return IPv6 addresses if a non-loopback IPv6 address is configured for the system.
all_matching	If used with v4_mapped, return all matching IPv6 and IPv4 addresses.
canonical_name	Determine the canonical name of the host specified in the query.
numeric_host	Host name should be treated as a numeric string defining an IPv4 or IPv6 address and no name resolution should be attempted.
numeric_service	Service name should be treated as a numeric string defining a port number and no name resolution should be attempted.
passive	Indicate that returned endpoint is intended for use as a locally bound socket endpoint.
v4_mapped	If the query protocol family is specified as IPv6, return IPv4-mapped IPv6 addresses on finding no IPv6 addresses.

Requirements

Header: `boost/asio/ip/resolver_query_base.hpp`

Convenience header: `boost/asio.hpp`

ip::resolver_query_base::address_configured

Only return IPv4 addresses if a non-loopback IPv4 address is configured for the system. Only return IPv6 addresses if a non-loopback IPv6 address is configured for the system.

```
static const flags address_configured = implementation_defined;
```

ip::resolver_query_base::all_matching

If used with v4_mapped, return all matching IPv6 and IPv4 addresses.

```
static const flags all_matching = implementation_defined;
```

ip::resolver_query_base::canonical_name

Determine the canonical name of the host specified in the query.

```
static const flags canonical_name = implementation_defined;
```

ip::resolver_query_base::flags

A bitmask type (C++ Std [lib.bitmask.types]).

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef unspecified flags;
```

Requirements

Header: `boost/asio/ip/resolver_query_base.hpp`

Convenience header: `boost/asio.hpp`

ip::resolver_query_base::numeric_host

Host name should be treated as a numeric string defining an IPv4 or IPv6 address and no name resolution should be attempted.

```
static const flags numeric_host = implementation_defined;
```

ip::resolver_query_base::numeric_service

Service name should be treated as a numeric string defining a port number and no name resolution should be attempted.

```
static const flags numeric_service = implementation_defined;
```

ip::resolver_query_base::passive

Indicate that returned endpoint is intended for use as a locally bound socket endpoint.

```
static const flags passive = implementation_defined;
```

ip::resolver_query_base::v4_mapped

If the query protocol family is specified as IPv6, return IPv4-mapped IPv6 addresses on finding no IPv6 addresses.

```
static const flags v4_mapped = implementation_defined;
```

ip::resolver_query_base::~resolver_query_base

Protected destructor to prevent deletion through this type.

```
~resolver_query_base();
```

ip::resolver_service

Default service implementation for a resolver.

```
template<
    typename InternetProtocol>
class resolver_service :
  public io_service::service
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
endpoint_type	The endpoint type.
implementation_type	The type of a resolver implementation.
iterator_type	The iterator type.
protocol_type	The protocol type.
query_type	The query type.

Member Functions

Name	Description
async_resolve	Asynchronously resolve a query to a list of entries. Asynchronously resolve an endpoint to a list of entries.
cancel	Cancel pending asynchronous operations.
construct	Construct a new resolver implementation.
destroy	Destroy a resolver implementation.
get_io_service	Get the io_service object that owns the service.
resolve	Resolve a query to a list of entries. Resolve an endpoint to a list of entries.
resolver_service	Construct a new resolver service for the specified io_service.

Data Members

Name	Description
id	The unique service identifier.

Requirements

Header: `boost/asio/ip/resolver_service.hpp`

Convenience header: `boost/asio.hpp`

ip::resolver_service::async_resolve

Asynchronously resolve a query to a list of entries.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ResolveHandler>
void async_resolve(
    implementation_type & impl,
    const query_type & query,
    ResolveHandler handler);
  » more...
```

Asynchronously resolve an endpoint to a list of entries.

```
template<
    typename ResolveHandler>
void async_resolve(
    implementation_type & impl,
    const endpoint_type & endpoint,
    ResolveHandler handler);
  » more...
```

ip::resolver_service::async_resolve (1 of 2 overloads)

Asynchronously resolve a query to a list of entries.

```
template<
    typename ResolveHandler>
void async_resolve(
    implementation_type & impl,
    const query_type & query,
    ResolveHandler handler);
```

ip::resolver_service::async_resolve (2 of 2 overloads)

Asynchronously resolve an endpoint to a list of entries.

```
template<
    typename ResolveHandler>
void async_resolve(
    implementation_type & impl,
    const endpoint_type & endpoint,
    ResolveHandler handler);
```

ip::resolver_service::cancel

Cancel pending asynchronous operations.

```
void cancel(
    implementation_type & impl);
```

ip::resolver_service::construct

Construct a new resolver implementation.

```
void construct(
    implementation_type & impl);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::resolver_service::destroy

Destroy a resolver implementation.

```
void destroy(
    implementation_type & impl);
```

ip::resolver_service::endpoint_type

The endpoint type.

```
typedef InternetProtocol::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/ip/resolver_service.hpp`

Convenience header: `boost/asio.hpp`

ip::resolver_service::get_io_service

Inherited from io_service.

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

ip::resolver_service::id

The unique service identifier.

```
static boost::asio::io_service::id id;
```

ip::resolver_service::implementation_type

The type of a resolver implementation.

```
typedef implementation_defined implementation_type;
```

Requirements

Header: `boost/asio/ip/resolver_service.hpp`

Convenience header: `boost/asio.hpp`

ip::resolver_service::iterator_type

The iterator type.

```
typedef basic_resolver_iterator< InternetProtocol > iterator_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
difference_type	The type used for the distance between two iterators.
iterator_category	The iterator category.
pointer	The type of the result of applying operator->() to the iterator.
reference	The type of the result of applying operator*() to the iterator.
value_type	The type of the value pointed to by the iterator.

Member Functions

Name	Description
basic_resolver_iterator	Default constructor creates an end iterator.
create	Create an iterator from an addrinfo list returned by getaddrinfo. Create an iterator from an endpoint, host name and service name.
operator *	Dereference an iterator.
operator++	Increment operator (prefix). Increment operator (postfix).
operator->	Dereference an iterator.

Friends

Name	Description
operator!=	Test two iterators for inequality.
operator==	Test two iterators for equality.

The `ip::basic_resolver_iterator` class template is used to define iterators over the results returned by a resolver.

The iterator's value_type, obtained when the iterator is dereferenced, is:

```
const basic_resolver_entry<InternetProtocol>
```

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/resolver_service.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::resolver_service::protocol_type

The protocol type.

```
typedef InternetProtocol protocol_type;
```

Requirements

Header: `boost/asio/ip/resolver_service.hpp`

Convenience header: `boost/asio.hpp`

ip::resolver_service::query_type

The query type.

```
typedef basic_resolver_query< InternetProtocol > query_type;
```

Types

Name	Description
flags	A bitmask type (C++ Std [lib.bitmask.types]).
protocol_type	The protocol type associated with the endpoint query.

Member Functions

Name	Description
basic_resolver_query	Construct with specified service name for any protocol. Construct with specified service name for a given protocol. Construct with specified host name and service name for any protocol. Construct with specified host name and service name for a given protocol.
hints	Get the hints associated with the query.
host_name	Get the host name associated with the query.
service_name	Get the service name associated with the query.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Data Members

Name	Description
address_configured	Only return IPv4 addresses if a non-loopback IPv4 address is configured for the system. Only return IPv6 addresses if a non-loopback IPv6 address is configured for the system.
all_matching	If used with v4_mapped, return all matching IPv6 and IPv4 addresses.
canonical_name	Determine the canonical name of the host specified in the query.
numeric_host	Host name should be treated as a numeric string defining an IPv4 or IPv6 address and no name resolution should be attempted.
numeric_service	Service name should be treated as a numeric string defining a port number and no name resolution should be attempted.
passive	Indicate that returned endpoint is intended for use as a locally bound socket endpoint.
v4_mapped	If the query protocol family is specified as IPv6, return IPv4-mapped IPv6 addresses on finding no IPv6 addresses.

The `ip::basic_resolver_query` class template describes a query that can be passed to a resolver.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/resolver_service.hpp`

Convenience header: `boost/asio.hpp`

ip::resolver_service::resolve

Resolve a query to a list of entries.

```
iterator_type resolve(
    implementation_type & impl,
    const query_type & query,
    boost::system::error_code & ec);
  » more...
```

Resolve an endpoint to a list of entries.

```
iterator_type resolve(
    implementation_type & impl,
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::resolver_service::resolve (1 of 2 overloads)

Resolve a query to a list of entries.

```
iterator_type resolve(
    implementation_type & impl,
    const query_type & query,
    boost::system::error_code & ec);
```

ip::resolver_service::resolve (2 of 2 overloads)

Resolve an endpoint to a list of entries.

```
iterator_type resolve(
    implementation_type & impl,
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
```

ip::resolver_service::resolver_service

Construct a new resolver service for the specified `io_service`.

```
resolver_service(
    boost::asio::io_service & io_service);
```

ip::tcp

Encapsulates the flags needed for TCP.

```
class tcp
```

Types

Name	Description
acceptor	The TCP acceptor type.
endpoint	The type of a TCP endpoint.
iostream	The TCP iostream type.
no_delay	Socket option for disabling the Nagle algorithm.
resolver	The TCP resolver type.
socket	The TCP socket type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
family	Obtain an identifier for the protocol family.
protocol	Obtain an identifier for the protocol.
type	Obtain an identifier for the type of the protocol.
v4	Construct to represent the IPv4 TCP protocol.
v6	Construct to represent the IPv6 TCP protocol.

Friends

Name	Description
operator!=	Compare two protocols for inequality.
operator==	Compare two protocols for equality.

The `ip::tcp` class contains flags necessary for TCP sockets.

Thread Safety

Distinct objects: Safe.

Shared objects: Safe.

Requirements

Header: `boost/asio/ip/tcp.hpp`

Convenience header: `boost/asio.hpp`

ip::tcp::acceptor

The TCP acceptor type.

```
typedef basic_socket_acceptor< tcp > acceptor;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of an acceptor.
native_type	(Deprecated: Use native_handle_type.) The native representation of an acceptor.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
accept	Accept a new connection. Accept a new connection and obtain the endpoint of the peer.
assign	Assigns an existing native acceptor to the acceptor.
async_accept	Start an asynchronous accept.
basic_socket_acceptor	Construct an acceptor without opening it. Construct an open acceptor. Construct an acceptor opened on the given endpoint. Construct a basic_socket_acceptor on an existing native acceptor. Move-construct a basic_socket_acceptor from another.
bind	Bind the acceptor to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the acceptor.
close	Close the acceptor.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the acceptor.
io_control	Perform an IO control command on the acceptor.
is_open	Determine whether the acceptor is open.
listen	Place the acceptor into the state where it will listen for new connections.
local_endpoint	Get the local endpoint of the acceptor.
native	(Deprecated: Use native_handle().) Get the native acceptor representation.
native_handle	Get the native acceptor representation.
native_non_blocking	Gets the non-blocking mode of the native acceptor implementation. Sets the non-blocking mode of the native acceptor implementation.
non_blocking	Gets the non-blocking mode of the acceptor. Sets the non-blocking mode of the acceptor.
open	Open the acceptor using the specified protocol.
operator=	Move-assign a basic_socket_acceptor from another.
set_option	Set an option on the acceptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_socket_acceptor` class template is used for accepting new socket connections.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Example

Opening a socket acceptor with the SO_REUSEADDR option enabled:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
boost::asio::ip::tcp::endpoint endpoint(boost::asio::ip::tcp::v4(), port);
acceptor.open(endpoint.protocol());
acceptor.set_option(boost::asio::ip::tcp::acceptor::reuse_address(true));
acceptor.bind(endpoint);
acceptor.listen();
```

Requirements

Header: `boost/asio/ip/tcp.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::tcp::endpoint

The type of a TCP endpoint.

```
typedef basic_endpoint< tcp > endpoint;
```

Types

Name	Description
data_type	The type of the endpoint structure. This type is dependent on the underlying implementation of the socket layer.
protocol_type	The protocol type associated with the endpoint.

Member Functions

Name	Description
address	Get the IP address associated with the endpoint. Set the IP address associated with the endpoint.
basic_endpoint	Default constructor. Construct an endpoint using a port number, specified in the host's byte order. The IP address will be the any address (i.e. INADDR_ANY or in6addr_any). This constructor would typically be used for accepting new connections. Construct an endpoint using a port number and an IP address. This constructor may be used for accepting connections on a specific interface or for making a connection to a remote endpoint. Copy constructor.
capacity	Get the capacity of the endpoint in the native type.
data	Get the underlying endpoint in the native type.
operator=	Assign from another endpoint.
port	Get the port associated with the endpoint. The port number is always in the host's byte order. Set the port associated with the endpoint. The port number is always in the host's byte order.
protocol	The protocol associated with the endpoint.
resize	Set the underlying size of the endpoint in the native type.
size	Get the underlying size of the endpoint in the native type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Friends

Name	Description
operator!=	Compare two endpoints for inequality.
operator<	Compare endpoints for ordering.
operator<=	Compare endpoints for ordering.
operator==	Compare two endpoints for equality.
operator>	Compare endpoints for ordering.
operator>=	Compare endpoints for ordering.

Related Functions

Name	Description
operator<<	Output an endpoint as a string.

The `ip::basic_endpoint` class template describes an endpoint that may be associated with a particular socket.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/tcp.hpp`

Convenience header: `boost/asio.hpp`

ip::tcp::family

Obtain an identifier for the protocol family.

```
int family() const;
```

ip::tcp::iostream

The TCP iostream type.

```
typedef basic_socket_iostream< tcp > iostream;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
duration_type	The duration type.
endpoint_type	The endpoint type.
time_type	The time type.

Member Functions

Name	Description
basic_socket_iostream	Construct a basic_socket_iostream without establishing a connection. Establish a connection to an endpoint corresponding to a resolver query.
close	Close the connection.
connect	Establish a connection to an endpoint corresponding to a resolver query.
error	Get the last error associated with the stream.
expires_at	Get the stream's expiry time as an absolute time. Set the stream's expiry time as an absolute time.
expires_from_now	Get the timer's expiry time relative to now. Set the stream's expiry time relative to now.
rdbuf	Return a pointer to the underlying streambuf.

Requirements

Header: `boost/asio/ip/tcp.hpp`

Convenience header: `boost/asio.hpp`

ip::tcp::no_delay

Socket option for disabling the Nagle algorithm.

```
typedef implementation_defined no_delay;
```

Implements the IPPROTO_TCP/TCP_NODELAY socket option.

Examples

Setting the option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::no_delay option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::tcp::no_delay option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/ip/tcp.hpp`

Convenience header: `boost/asio.hpp`

ip::tcp::operator!=

Compare two protocols for inequality.

```
friend bool operator!=(
    const tcp & p1,
    const tcp & p2);
```

Requirements

Header: `boost/asio/ip/tcp.hpp`

Convenience header: `boost/asio.hpp`

ip::tcp::operator==

Compare two protocols for equality.

```
friend bool operator==(
    const tcp & p1,
    const tcp & p2);
```

Requirements

Header: `boost/asio/ip/tcp.hpp`

Convenience header: `boost/asio.hpp`

ip::tcp::protocol

Obtain an identifier for the protocol.

```
int protocol() const;
```

ip::tcp::resolver

The TCP resolver type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef basic_resolver< tcp > resolver;
```

Types

Name	Description
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
iterator	The iterator type.
protocol_type	The protocol type.
query	The query type.
service_type	The type of the service that will be used to provide I/O operations.

Member Functions

Name	Description
async_resolve	Asynchronously perform forward resolution of a query to a list of entries. Asynchronously perform reverse resolution of an endpoint to a list of entries.
basic_resolver	Constructor.
cancel	Cancel any asynchronous operations that are waiting on the resolver.
get_io_service	Get the io_service associated with the object.
resolve	Perform forward resolution of a query to a list of entries. Perform reverse resolution of an endpoint to a list of entries.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `ip::basic_resolver` class template provides the ability to resolve a query to a list of endpoints.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/tcp.hpp`

Convenience header: `boost/asio.hpp`

ip::tcp::socket

The TCP socket type.

```
typedef basic_stream_socket< tcp > socket;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
async_read_some	Start an asynchronous read.
async_receive	Start an asynchronous receive.
async_send	Start an asynchronous send.
async_write_some	Start an asynchronous write.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_stream_socket	Construct a basic_stream_socket without opening it. Construct and open a basic_stream_socket. Construct a basic_stream_socket, opening it and binding it to the given local endpoint. Construct a basic_stream_socket on an existing native socket. Move-construct a basic_stream_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_stream_socket from another.
read_some	Read some data from the socket.
receive	Receive some data on the socket. Receive some data on a connected socket.
remote_endpoint	Get the remote endpoint of the socket.
send	Send some data on the socket.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.
write_some	Write some data to the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_stream_socket` class template provides asynchronous and blocking stream-oriented socket functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/tcp.hpp`

Convenience header: `boost/asio.hpp`

ip::tcp::type

Obtain an identifier for the type of the protocol.

```
int type() const;
```

ip::tcp::v4

Construct to represent the IPv4 TCP protocol.

```
static tcp v4();
```

ip::tcp::v6

Construct to represent the IPv6 TCP protocol.

```
static tcp v6();
```

ip::udp

Encapsulates the flags needed for UDP.

```
class udp
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
endpoint	The type of a UDP endpoint.
resolver	The UDP resolver type.
socket	The UDP socket type.

Member Functions

Name	Description
family	Obtain an identifier for the protocol family.
protocol	Obtain an identifier for the protocol.
type	Obtain an identifier for the type of the protocol.
v4	Construct to represent the IPv4 UDP protocol.
v6	Construct to represent the IPv6 UDP protocol.

Friends

Name	Description
operator!=	Compare two protocols for inequality.
operator==	Compare two protocols for equality.

The `ip::udp` class contains flags necessary for UDP sockets.

Thread Safety

Distinct objects: Safe.

Shared objects: Safe.

Requirements

Header: `boost/asio/ip/udp.hpp`

Convenience header: `boost/asio.hpp`

ip::udp::endpoint

The type of a UDP endpoint.

```
typedef basic_endpoint< udp > endpoint;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
data_type	The type of the endpoint structure. This type is dependent on the underlying implementation of the socket layer.
protocol_type	The protocol type associated with the endpoint.

Member Functions

Name	Description
address	Get the IP address associated with the endpoint. Set the IP address associated with the endpoint.
basic_endpoint	Default constructor. Construct an endpoint using a port number, specified in the host's byte order. The IP address will be the any address (i.e. INADDR_ANY or in6addr_any). This constructor would typically be used for accepting new connections. Construct an endpoint using a port number and an IP address. This constructor may be used for accepting connections on a specific interface or for making a connection to a remote endpoint. Copy constructor.
capacity	Get the capacity of the endpoint in the native type.
data	Get the underlying endpoint in the native type.
operator=	Assign from another endpoint.
port	Get the port associated with the endpoint. The port number is always in the host's byte order. Set the port associated with the endpoint. The port number is always in the host's byte order.
protocol	The protocol associated with the endpoint.
resize	Set the underlying size of the endpoint in the native type.
size	Get the underlying size of the endpoint in the native type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Friends

Name	Description
operator!=	Compare two endpoints for inequality.
operator<	Compare endpoints for ordering.
operator<=	Compare endpoints for ordering.
operator==	Compare two endpoints for equality.
operator>	Compare endpoints for ordering.
operator>=	Compare endpoints for ordering.

Related Functions

Name	Description
operator<<	Output an endpoint as a string.

The `ip::basic_endpoint` class template describes an endpoint that may be associated with a particular socket.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/udp.hpp`

Convenience header: `boost/asio.hpp`

ip::udp::family

Obtain an identifier for the protocol family.

```
int family() const;
```

ip::udp::operator!=

Compare two protocols for inequality.

```
friend bool operator!=(
    const udp & p1,
    const udp & p2);
```

Requirements

Header: `boost/asio/ip/udp.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::udp::operator==

Compare two protocols for equality.

```
friend bool operator==(
    const udp & p1,
    const udp & p2);
```

Requirements

Header: `boost/asio/ip/udp.hpp`

Convenience header: `boost/asio.hpp`

ip::udp::protocol

Obtain an identifier for the protocol.

```
int protocol() const;
```

ip::udp::resolver

The UDP resolver type.

```
typedef basic_resolver< udp > resolver;
```

Types

Name	Description
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
iterator	The iterator type.
protocol_type	The protocol type.
query	The query type.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
async_resolve	Asynchronously perform forward resolution of a query to a list of entries. Asynchronously perform reverse resolution of an endpoint to a list of entries.
basic_resolver	Constructor.
cancel	Cancel any asynchronous operations that are waiting on the resolver.
get_io_service	Get the io_service associated with the object.
resolve	Perform forward resolution of a query to a list of entries. Perform reverse resolution of an endpoint to a list of entries.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `ip::basic_resolver` class template provides the ability to resolve a query to a list of endpoints.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/udp.hpp`

Convenience header: `boost/asio.hpp`

ip::udp::socket

The UDP socket type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef basic_datagram_socket< udp > socket;
```

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
async_receive	Start an asynchronous receive on a connected socket.
async_receive_from	Start an asynchronous receive.
async_send	Start an asynchronous send on a connected socket.
async_send_to	Start an asynchronous send.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_datagram_socket	Construct a basic_datagram_socket without opening it. Construct and open a basic_datagram_socket. Construct a basic_datagram_socket, opening it and binding it to the given local endpoint. Construct a basic_datagram_socket on an existing native socket. Move-construct a basic_datagram_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_datagram_socket from another.
receive	Receive some data on a connected socket.
receive_from	Receive a datagram with the endpoint of the sender.
remote_endpoint	Get the remote endpoint of the socket.
send	Send some data on a connected socket.
send_to	Send a datagram to the specified endpoint.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_datagram_socket` class template provides asynchronous and blocking datagram-oriented socket functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/ip/udp.hpp`

Convenience header: `boost/asio.hpp`

ip::udp::type

Obtain an identifier for the type of the protocol.

```
int type() const;
```

ip::udp::v4

Construct to represent the IPv4 UDP protocol.

```
static udp v4();
```

ip::udp::v6

Construct to represent the IPv6 UDP protocol.

```
static udp v6();
```

ip::unicast::hops

Socket option for time-to-live associated with outgoing unicast packets.

```
typedef implementation_defined hops;
```

Implements the IPPROTO_IP/IP_UNICAST_TTL socket option.

Examples

Setting the option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::ip::unicast::hops option(4);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::ip::unicast::hops option;
socket.get_option(option);
int ttl = option.value();
```

Requirements

Header: `boost/asio/ip/unicast.hpp`

Convenience header: `boost/asio.hpp`

ip::v6_only

Socket option for determining whether an IPv6 socket supports IPv6 communication only.

```
typedef implementation_defined v6_only;
```

Implements the IPPROTO_IPV6/IP_V6ONLY socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::v6_only option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::ip::v6_only option;
socket.get_option(option);
bool v6_only = option.value();
```

Requirements

Header: `boost/asio/ip/v6_only.hpp`

Convenience header: `boost/asio.hpp`

is_match_condition

Type trait used to determine whether a type can be used as a match condition function with read_until and async_read_until.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename T>
struct is_match_condition
```

Data Members

Name	Description
value	The value member is true if the type may be used as a match condition.

Requirements

Header: `boost/asio/read_until.hpp`

Convenience header: `boost/asio.hpp`

is_match_condition::value

The value member is true if the type may be used as a match condition.

```
static const bool value;
```

is_read_buffered

The `is_read_buffered` class is a traits class that may be used to determine whether a stream type supports buffering of read data.

```
template<
    typename Stream>
class is_read_buffered
```

Data Members

Name	Description
value	The value member is true only if the Stream type supports buffering of read data.

Requirements

Header: `boost/asio/is_read_buffered.hpp`

Convenience header: `boost/asio.hpp`

is_read_buffered::value

The value member is true only if the Stream type supports buffering of read data.

```
static const bool value;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

is_write_buffered

The `is_write_buffered` class is a traits class that may be used to determine whether a stream type supports buffering of written data.

```
template<
    typename Stream>
class is_write_buffered
```

Data Members

Name	Description
value	The value member is true only if the Stream type supports buffering of written data.

Requirements

Header: `boost/asio/is_write_buffered.hpp`

Convenience header: `boost/asio.hpp`

is_write_buffered::value

The value member is true only if the Stream type supports buffering of written data.

```
static const bool value;
```

local::basic_endpoint

Describes an endpoint for a UNIX socket.

```
template<
    typename Protocol>
class basic_endpoint
```

Types

Name	Description
data_type	The type of the endpoint structure. This type is dependent on the underlying implementation of the socket layer.
protocol_type	The protocol type associated with the endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
basic_endpoint	Default constructor. Construct an endpoint using the specified path name. Copy constructor.
capacity	Get the capacity of the endpoint in the native type.
data	Get the underlying endpoint in the native type.
operator=	Assign from another endpoint.
path	Get the path associated with the endpoint. Set the path associated with the endpoint.
protocol	The protocol associated with the endpoint.
resize	Set the underlying size of the endpoint in the native type.
size	Get the underlying size of the endpoint in the native type.

Friends

Name	Description
operator!=	Compare two endpoints for inequality.
operator<	Compare endpoints for ordering.
operator<=	Compare endpoints for ordering.
operator==	Compare two endpoints for equality.
operator>	Compare endpoints for ordering.
operator>=	Compare endpoints for ordering.

Related Functions

Name	Description
operator<<	Output an endpoint as a string.

The `local::basic_endpoint` class template describes an endpoint that may be associated with a particular UNIX socket.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/local/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

local::basic_endpoint::basic_endpoint

Default constructor.

```
basic_endpoint();
  » more...
```

Construct an endpoint using the specified path name.

```
basic_endpoint(
    const char * path_name);
  » more...

basic_endpoint(
    const std::string & path_name);
  » more...
```

Copy constructor.

```
basic_endpoint(
    const basic_endpoint & other);
  » more...
```

local::basic_endpoint::basic_endpoint (1 of 4 overloads)

Default constructor.

```
basic_endpoint();
```

local::basic_endpoint::basic_endpoint (2 of 4 overloads)

Construct an endpoint using the specified path name.

```
basic_endpoint(
    const char * path_name);
```

local::basic_endpoint::basic_endpoint (3 of 4 overloads)

Construct an endpoint using the specified path name.

```
basic_endpoint(
    const std::string & path_name);
```

local::basic_endpoint::basic_endpoint (4 of 4 overloads)

Copy constructor.

```
basic_endpoint(
    const basic_endpoint & other);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

local::basic_endpoint::capacity

Get the capacity of the endpoint in the native type.

```
std::size_t capacity() const;
```

local::basic_endpoint::data

Get the underlying endpoint in the native type.

```
data_type * data();
  » more...

const data_type * data() const;
  » more...
```

local::basic_endpoint::data (1 of 2 overloads)

Get the underlying endpoint in the native type.

```
data_type * data();
```

local::basic_endpoint::data (2 of 2 overloads)

Get the underlying endpoint in the native type.

```
const data_type * data() const;
```

local::basic_endpoint::data_type

The type of the endpoint structure. This type is dependent on the underlying implementation of the socket layer.

```
typedef implementation_defined data_type;
```

Requirements

Header: `boost/asio/local/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

local::basic_endpoint::operator!=

Compare two endpoints for inequality.

```
friend bool operator!=(
    const basic_endpoint< Protocol > & e1,
    const basic_endpoint< Protocol > & e2);
```

Requirements

Header: `boost/asio/local/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

local::basic_endpoint::operator<

Compare endpoints for ordering.

```
friend bool operator<(
    const basic_endpoint< Protocol > & e1,
    const basic_endpoint< Protocol > & e2);
```

Requirements

Header: `boost/asio/local/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

local::basic_endpoint::operator<<

Output an endpoint as a string.

```
std::basic_ostream< Elem, Traits > & operator<<(
    std::basic_ostream< Elem, Traits > & os,
    const basic_endpoint< Protocol > & endpoint);
```

Used to output a human-readable string for a specified endpoint.

Parameters

os	The output stream to which the string will be written.
endpoint	The endpoint to be written.

Return Value

The output stream.

local::basic_endpoint::operator<=

Compare endpoints for ordering.

```
friend bool operator<=(
    const basic_endpoint< Protocol > & e1,
    const basic_endpoint< Protocol > & e2);
```

Requirements

Header: `boost/asio/local/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

local::basic_endpoint::operator=

Assign from another endpoint.

```
basic_endpoint & operator=(
    const basic_endpoint & other);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

local::basic_endpoint::operator==

Compare two endpoints for equality.

```
friend bool operator==(
    const basic_endpoint< Protocol > & e1,
    const basic_endpoint< Protocol > & e2);
```

Requirements

Header: `boost/asio/local/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

local::basic_endpoint::operator>

Compare endpoints for ordering.

```
friend bool operator>(
    const basic_endpoint< Protocol > & e1,
    const basic_endpoint< Protocol > & e2);
```

Requirements

Header: `boost/asio/local/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

local::basic_endpoint::operator>=

Compare endpoints for ordering.

```
friend bool operator>=(
    const basic_endpoint< Protocol > & e1,
    const basic_endpoint< Protocol > & e2);
```

Requirements

Header: `boost/asio/local/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

local::basic_endpoint::path

Get the path associated with the endpoint.

```
std::string path() const;
  » more...
```

Set the path associated with the endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void path(
    const char * p);
  » more...

void path(
    const std::string & p);
  » more...
```

local::basic_endpoint::path (1 of 3 overloads)

Get the path associated with the endpoint.

```
std::string path() const;
```

local::basic_endpoint::path (2 of 3 overloads)

Set the path associated with the endpoint.

```
void path(
    const char * p);
```

local::basic_endpoint::path (3 of 3 overloads)

Set the path associated with the endpoint.

```
void path(
    const std::string & p);
```

local::basic_endpoint::protocol

The protocol associated with the endpoint.

```
protocol_type protocol() const;
```

local::basic_endpoint::protocol_type

The protocol type associated with the endpoint.

```
typedef Protocol protocol_type;
```

Requirements

Header: `boost/asio/local/basic_endpoint.hpp`

Convenience header: `boost/asio.hpp`

local::basic_endpoint::resize

Set the underlying size of the endpoint in the native type.

```
void resize(
    std::size_t new_size);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

local::basic_endpoint::size

Get the underlying size of the endpoint in the native type.

```
std::size_t size() const;
```

local::connect_pair

Create a pair of connected sockets.

```
template<
    typename Protocol,
    typename SocketService1,
    typename SocketService2>
void connect_pair(
    basic_socket< Protocol, SocketService1 > & socket1,
    basic_socket< Protocol, SocketService2 > & socket2);
  » more...

template<
    typename Protocol,
    typename SocketService1,
    typename SocketService2>
boost::system::error_code connect_pair(
    basic_socket< Protocol, SocketService1 > & socket1,
    basic_socket< Protocol, SocketService2 > & socket2,
    boost::system::error_code & ec);
  » more...
```

Requirements

Header: `boost/asio/local/connect_pair.hpp`

Convenience header: `boost/asio.hpp`

local::connect_pair (1 of 2 overloads)

Create a pair of connected sockets.

```
template<
    typename Protocol,
    typename SocketService1,
    typename SocketService2>
void connect_pair(
    basic_socket< Protocol, SocketService1 > & socket1,
    basic_socket< Protocol, SocketService2 > & socket2);
```

local::connect_pair (2 of 2 overloads)

Create a pair of connected sockets.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename Protocol,
    typename SocketService1,
    typename SocketService2>
boost::system::error_code connect_pair(
    basic_socket< Protocol, SocketService1 > & socket1,
    basic_socket< Protocol, SocketService2 > & socket2,
    boost::system::error_code & ec);
```

local::datagram_protocol

Encapsulates the flags needed for datagram-oriented UNIX sockets.

```
class datagram_protocol
```

Types

Name	Description
endpoint	The type of a UNIX domain endpoint.
socket	The UNIX domain socket type.

Member Functions

Name	Description
family	Obtain an identifier for the protocol family.
protocol	Obtain an identifier for the protocol.
type	Obtain an identifier for the type of the protocol.

The `local::datagram_protocol` class contains flags necessary for datagram-oriented UNIX domain sockets.

Thread Safety

Distinct objects: Safe.

Shared objects: Safe.

Requirements

Header: `boost/asio/local/datagram_protocol.hpp`

Convenience header: `boost/asio.hpp`

local::datagram_protocol::endpoint

The type of a UNIX domain endpoint.

```
typedef basic_endpoint< datagram_protocol > endpoint;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
data_type	The type of the endpoint structure. This type is dependent on the underlying implementation of the socket layer.
protocol_type	The protocol type associated with the endpoint.

Member Functions

Name	Description
basic_endpoint	Default constructor. Construct an endpoint using the specified path name. Copy constructor.
capacity	Get the capacity of the endpoint in the native type.
data	Get the underlying endpoint in the native type.
operator=	Assign from another endpoint.
path	Get the path associated with the endpoint. Set the path associated with the endpoint.
protocol	The protocol associated with the endpoint.
resize	Set the underlying size of the endpoint in the native type.
size	Get the underlying size of the endpoint in the native type.

Friends

Name	Description
operator!=	Compare two endpoints for inequality.
operator<	Compare endpoints for ordering.
operator<=	Compare endpoints for ordering.
operator==	Compare two endpoints for equality.
operator>	Compare endpoints for ordering.
operator>=	Compare endpoints for ordering.

Related Functions

Name	Description
operator<<	Output an endpoint as a string.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The `local::basic_endpoint` class template describes an endpoint that may be associated with a particular UNIX socket.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/local/datagram_protocol.hpp`

Convenience header: `boost/asio.hpp`

local::datagram_protocol::family

Obtain an identifier for the protocol family.

```
int family() const;
```

local::datagram_protocol::protocol

Obtain an identifier for the protocol.

```
int protocol() const;
```

local::datagram_protocol::socket

The UNIX domain socket type.

```
typedef basic_datagram_socket< datagram_protocol > socket;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
async_receive	Start an asynchronous receive on a connected socket.
async_receive_from	Start an asynchronous receive.
async_send	Start an asynchronous send on a connected socket.
async_send_to	Start an asynchronous send.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_datagram_socket	Construct a basic_datagram_socket without opening it. Construct and open a basic_datagram_socket. Construct a basic_datagram_socket, opening it and binding it to the given local endpoint. Construct a basic_datagram_socket on an existing native socket. Move-construct a basic_datagram_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_datagram_socket from another.
receive	Receive some data on a connected socket.
receive_from	Receive a datagram with the endpoint of the sender.
remote_endpoint	Get the remote endpoint of the socket.
send	Send some data on a connected socket.
send_to	Send a datagram to the specified endpoint.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_datagram_socket` class template provides asynchronous and blocking datagram-oriented socket functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/local/datagram_protocol.hpp`

Convenience header: `boost/asio.hpp`

local::datagram_protocol::type

Obtain an identifier for the type of the protocol.

```
int type() const;
```

local::stream_protocol

Encapsulates the flags needed for stream-oriented UNIX sockets.

```
class stream_protocol
```

Types

Name	Description
acceptor	The UNIX domain acceptor type.
endpoint	The type of a UNIX domain endpoint.
iostream	The UNIX domain iostream type.
socket	The UNIX domain socket type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
family	Obtain an identifier for the protocol family.
protocol	Obtain an identifier for the protocol.
type	Obtain an identifier for the type of the protocol.

The `local::stream_protocol` class contains flags necessary for stream-oriented UNIX domain sockets.

Thread Safety

Distinct objects: Safe.

Shared objects: Safe.

Requirements

Header: `boost/asio/local/stream_protocol.hpp`

Convenience header: `boost/asio.hpp`

local::stream_protocol::acceptor

The UNIX domain acceptor type.

```
typedef basic_socket_acceptor< stream_protocol > acceptor;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of an acceptor.
native_type	(Deprecated: Use native_handle_type.) The native representation of an acceptor.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
accept	Accept a new connection. Accept a new connection and obtain the endpoint of the peer.
assign	Assigns an existing native acceptor to the acceptor.
async_accept	Start an asynchronous accept.
basic_socket_acceptor	Construct an acceptor without opening it. Construct an open acceptor. Construct an acceptor opened on the given endpoint. Construct a basic_socket_acceptor on an existing native acceptor. Move-construct a basic_socket_acceptor from another.
bind	Bind the acceptor to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the acceptor.
close	Close the acceptor.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the acceptor.
io_control	Perform an IO control command on the acceptor.
is_open	Determine whether the acceptor is open.
listen	Place the acceptor into the state where it will listen for new connections.
local_endpoint	Get the local endpoint of the acceptor.
native	(Deprecated: Use native_handle().) Get the native acceptor representation.
native_handle	Get the native acceptor representation.
native_non_blocking	Gets the non-blocking mode of the native acceptor implementation. Sets the non-blocking mode of the native acceptor implementation.
non_blocking	Gets the non-blocking mode of the acceptor. Sets the non-blocking mode of the acceptor.
open	Open the acceptor using the specified protocol.
operator=	Move-assign a basic_socket_acceptor from another.
set_option	Set an option on the acceptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_socket_acceptor` class template is used for accepting new socket connections.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Example

Opening a socket acceptor with the SO_REUSEADDR option enabled:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
boost::asio::ip::tcp::endpoint endpoint(boost::asio::ip::tcp::v4(), port);
acceptor.open(endpoint.protocol());
acceptor.set_option(boost::asio::ip::tcp::acceptor::reuse_address(true));
acceptor.bind(endpoint);
acceptor.listen();
```

Requirements

Header: `boost/asio/local/stream_protocol.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

local::stream_protocol::endpoint

The type of a UNIX domain endpoint.

```
typedef basic_endpoint< stream_protocol > endpoint;
```

Types

Name	Description
data_type	The type of the endpoint structure. This type is dependent on the underlying implementation of the socket layer.
protocol_type	The protocol type associated with the endpoint.

Member Functions

Name	Description
basic_endpoint	Default constructor. Construct an endpoint using the specified path name. Copy constructor.
capacity	Get the capacity of the endpoint in the native type.
data	Get the underlying endpoint in the native type.
operator=	Assign from another endpoint.
path	Get the path associated with the endpoint. Set the path associated with the endpoint.
protocol	The protocol associated with the endpoint.
resize	Set the underlying size of the endpoint in the native type.
size	Get the underlying size of the endpoint in the native type.

Friends

Name	Description
operator!=	Compare two endpoints for inequality.
operator<	Compare endpoints for ordering.
operator<=	Compare endpoints for ordering.
operator==	Compare two endpoints for equality.
operator>	Compare endpoints for ordering.
operator>=	Compare endpoints for ordering.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Related Functions

Name	Description
operator<<	Output an endpoint as a string.

The `local::basic_endpoint` class template describes an endpoint that may be associated with a particular UNIX socket.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/local/stream_protocol.hpp`

Convenience header: `boost/asio.hpp`

local::stream_protocol::family

Obtain an identifier for the protocol family.

```
int family() const;
```

local::stream_protocol::iostream

The UNIX domain iostream type.

```
typedef basic_socket_iostream< stream_protocol > iostream;
```

Types

Name	Description
duration_type	The duration type.
endpoint_type	The endpoint type.
time_type	The time type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
basic_socket_iostream	Construct a basic_socket_iostream without establishing a connection. Establish a connection to an endpoint corresponding to a resolver query.
close	Close the connection.
connect	Establish a connection to an endpoint corresponding to a resolver query.
error	Get the last error associated with the stream.
expires_at	Get the stream's expiry time as an absolute time. Set the stream's expiry time as an absolute time.
expires_from_now	Get the timer's expiry time relative to now. Set the stream's expiry time relative to now.
rdbuf	Return a pointer to the underlying streambuf.

Requirements

Header: `boost/asio/local/stream_protocol.hpp`

Convenience header: `boost/asio.hpp`

local::stream_protocol::protocol

Obtain an identifier for the protocol.

```
int protocol() const;
```

local::stream_protocol::socket

The UNIX domain socket type.

```
typedef basic_stream_socket< stream_protocol > socket;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
endpoint_type	The endpoint type.
implementation_type	The underlying implementation type of I/O object.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
lowest_layer_type	A basic_socket is always the lowest layer.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
native_handle_type	The native representation of a socket.
native_type	(Deprecated: Use native_handle_type.) The native representation of a socket.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
protocol_type	The protocol type.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
service_type	The type of the service that will be used to provide I/O operations.
shutdown_type	Different ways a socket may be shutdown.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to the socket.
async_connect	Start an asynchronous connect.
async_read_some	Start an asynchronous read.
async_receive	Start an asynchronous receive.
async_send	Start an asynchronous send.
async_write_some	Start an asynchronous write.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
basic_stream_socket	Construct a basic_stream_socket without opening it. Construct and open a basic_stream_socket. Construct a basic_stream_socket, opening it and binding it to the given local endpoint. Construct a basic_stream_socket on an existing native socket. Move-construct a basic_stream_socket from another.
bind	Bind the socket to the given local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close the socket.
connect	Connect the socket to the specified endpoint.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the socket.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint of the socket.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native socket representation.
native_handle	Get the native socket representation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open the socket using the specified protocol.
operator=	Move-assign a basic_stream_socket from another.
read_some	Read some data from the socket.
receive	Receive some data on the socket. Receive some data on a connected socket.
remote_endpoint	Get the remote endpoint of the socket.
send	Send some data on the socket.
set_option	Set an option on the socket.
shutdown	Disable sends or receives on the socket.
write_some	Write some data to the socket.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_stream_socket` class template provides asynchronous and blocking stream-oriented socket functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/local/stream_protocol.hpp`

Convenience header: `boost/asio.hpp`

local::stream_protocol::type

Obtain an identifier for the type of the protocol.

```
int type() const;
```

mutable_buffer

Holds a buffer that can be modified.

```
class mutable_buffer
```

Member Functions

Name	Description
mutable_buffer	Construct an empty buffer. Construct a buffer to represent a given memory range.

Related Functions

Name	Description
operator+	Create a new modifiable buffer that is offset from the start of another.

The `mutable_buffer` class provides a safe representation of a buffer that can be modified. It does not own the underlying data, and so is cheap to copy or assign.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Accessing Buffer Contents

The contents of a buffer may be accessed using the `buffer_size` and `buffer_cast` functions:

```
boost::asio::mutable_buffer b1 = ...;
std::size_t s1 = boost::asio::buffer_size(b1);
unsigned char* p1 = boost::asio::buffer_cast<unsigned char*>(b1);
```

The `boost::asio::buffer_cast` function permits violations of type safety, so uses of it in application code should be carefully considered.

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

mutable_buffer::mutable_buffer

Construct an empty buffer.

```
mutable_buffer();
  » more...
```

Construct a buffer to represent a given memory range.

```
mutable_buffer(
    void * data,
    std::size_t size);
  » more...
```

mutable_buffer::mutable_buffer (1 of 2 overloads)

Construct an empty buffer.

```
mutable_buffer();
```

mutable_buffer::mutable_buffer (2 of 2 overloads)

Construct a buffer to represent a given memory range.

```
mutable_buffer(
    void * data,
    std::size_t size);
```

mutable_buffer::operator+

Create a new modifiable buffer that is offset from the start of another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
mutable_buffer operator+(
    const mutable_buffer & b,
    std::size_t start);
  » more...

mutable_buffer operator+(
    std::size_t start,
    const mutable_buffer & b);
  » more...
```

mutable_buffer::operator+ (1 of 2 overloads)

Create a new modifiable buffer that is offset from the start of another.

```
mutable_buffer operator+(
    const mutable_buffer & b,
    std::size_t start);
```

mutable_buffer::operator+ (2 of 2 overloads)

Create a new modifiable buffer that is offset from the start of another.

```
mutable_buffer operator+(
    std::size_t start,
    const mutable_buffer & b);
```

mutable_buffers_1

Adapts a single modifiable buffer so that it meets the requirements of the MutableBufferSequence concept.

```
class mutable_buffers_1 :
  public mutable_buffer
```

Types

Name	Description
const_iterator	A random-access iterator type that may be used to read elements.
value_type	The type for each element in the list of buffers.

Member Functions

Name	Description
begin	Get a random-access iterator to the first element.
end	Get a random-access iterator for one past the last element.
mutable_buffers_1	Construct to represent a given memory range. Construct to represent a single modifiable buffer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Related Functions

Name	Description
operator+	Create a new modifiable buffer that is offset from the start of another.

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

mutable_buffers_1::begin

Get a random-access iterator to the first element.

```
const_iterator begin() const;
```

mutable_buffers_1::const_iterator

A random-access iterator type that may be used to read elements.

```
typedef const mutable_buffer * const_iterator;
```

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

mutable_buffers_1::end

Get a random-access iterator for one past the last element.

```
const_iterator end() const;
```

mutable_buffers_1::mutable_buffers_1

Construct to represent a given memory range.

```
mutable_buffers_1(
    void * data,
    std::size_t size);
  » more...
```

Construct to represent a single modifiable buffer.

```
explicit mutable_buffers_1(
    const mutable_buffer & b);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

mutable_buffers_1::mutable_buffers_1 (1 of 2 overloads)

Construct to represent a given memory range.

```
mutable_buffers_1(
    void * data,
    std::size_t size);
```

mutable_buffers_1::mutable_buffers_1 (2 of 2 overloads)

Construct to represent a single modifiable buffer.

```
mutable_buffers_1(
    const mutable_buffer & b);
```

mutable_buffers_1::operator+

Create a new modifiable buffer that is offset from the start of another.

```
mutable_buffer operator+(
    const mutable_buffer & b,
    std::size_t start);
  » more...

mutable_buffer operator+(
    std::size_t start,
    const mutable_buffer & b);
  » more...
```

mutable_buffers_1::operator+ (1 of 2 overloads)

Inherited from mutable_buffer.

Create a new modifiable buffer that is offset from the start of another.

```
mutable_buffer operator+(
    const mutable_buffer & b,
    std::size_t start);
```

mutable_buffers_1::operator+ (2 of 2 overloads)

Inherited from mutable_buffer.

Create a new modifiable buffer that is offset from the start of another.

```
mutable_buffer operator+(
    std::size_t start,
    const mutable_buffer & b);
```

mutable_buffers_1::value_type

The type for each element in the list of buffers.

```
typedef mutable_buffer value_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
mutable_buffer	Construct an empty buffer. Construct a buffer to represent a given memory range.

Related Functions

Name	Description
operator+	Create a new modifiable buffer that is offset from the start of another.

The `mutable_buffer` class provides a safe representation of a buffer that can be modified. It does not own the underlying data, and so is cheap to copy or assign.

Accessing Buffer Contents

The contents of a buffer may be accessed using the `buffer_size` and `buffer_cast` functions:

```
boost::asio::mutable_buffer b1 = ...;
std::size_t s1 = boost::asio::buffer_size(b1);
unsigned char* p1 = boost::asio::buffer_cast<unsigned char*>(b1);
```

The `boost::asio::buffer_cast` function permits violations of type safety, so uses of it in application code should be carefully considered.

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

null_buffers

An implementation of both the ConstBufferSequence and MutableBufferSequence concepts to represent a null buffer sequence.

```
class null_buffers
```

Types

Name	Description
const_iterator	A random-access iterator type that may be used to read elements.
value_type	The type for each element in the list of buffers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
begin	Get a random-access iterator to the first element.
end	Get a random-access iterator for one past the last element.

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

null_buffers::begin

Get a random-access iterator to the first element.

```
const_iterator begin() const;
```

null_buffers::const_iterator

A random-access iterator type that may be used to read elements.

```
typedef const mutable_buffer * const_iterator;
```

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

null_buffers::end

Get a random-access iterator for one past the last element.

```
const_iterator end() const;
```

null_buffers::value_type

The type for each element in the list of buffers.

```
typedef mutable_buffer value_type;
```

Member Functions

Name	Description
mutable_buffer	Construct an empty buffer. Construct a buffer to represent a given memory range.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Related Functions

Name	Description
operator+	Create a new modifiable buffer that is offset from the start of another.

The `mutable_buffer` class provides a safe representation of a buffer that can be modified. It does not own the underlying data, and so is cheap to copy or assign.

Accessing Buffer Contents

The contents of a buffer may be accessed using the `buffer_size` and `buffer_cast` functions:

```
boost::asio::mutable_buffer b1 = ...;
std::size_t s1 = boost::asio::buffer_size(b1);
unsigned char* p1 = boost::asio::buffer_cast<unsigned char*>(b1);
```

The `boost::asio::buffer_cast` function permits violations of type safety, so uses of it in application code should be carefully considered.

Requirements

Header: `boost/asio/buffer.hpp`

Convenience header: `boost/asio.hpp`

placeholders::bytes_transferred

An argument placeholder, for use with boost::bind(), that corresponds to the bytes_transferred argument of a handler for asynchronous functions such as `boost::asio::basic_stream_socket::async_write_some` or `boost::asio::async_write`.

```
unspecified bytes_transferred;
```

Requirements

Header: `boost/asio/placeholders.hpp`

Convenience header: `boost/asio.hpp`

placeholders::error

An argument placeholder, for use with boost::bind(), that corresponds to the error argument of a handler for any of the asynchronous functions.

```
unspecified error;
```

Requirements

Header: `boost/asio/placeholders.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

placeholders::iterator

An argument placeholder, for use with boost::bind(), that corresponds to the iterator argument of a handler for asynchronous functions such as boost::asio::basic_resolver::async_resolve.

```
unspecified iterator;
```

Requirements

Header: `boost/asio/placeholders.hpp`

Convenience header: `boost/asio.hpp`

placeholders::signal_number

An argument placeholder, for use with boost::bind(), that corresponds to the signal_number argument of a handler for asynchronous functions such as `boost::asio::signal_set::async_wait`.

```
unspecified signal_number;
```

Requirements

Header: `boost/asio/placeholders.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_descriptor

Provides POSIX descriptor functionality.

```
template<
    typename DescriptorService>
class basic_descriptor :
  public basic_io_object< DescriptorService >,
  public posix::descriptor_base
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
bytes_readable	IO control command to get the amount of data that can be read without blocking.
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_descriptor is always the lowest layer.
native_handle_type	The native representation of a descriptor.
native_type	(Deprecated: Use native_handle_type.) The native representation of a descriptor.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the descriptor.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native descriptor to the descriptor.
basic_descriptor	Construct a basic_descriptor without opening it. Construct a basic_descriptor on an existing native descriptor. Move-construct a basic_descriptor from another.
cancel	Cancel all asynchronous operations associated with the descriptor.
close	Close the descriptor.
get_io_service	Get the io_service associated with the object.
io_control	Perform an IO control command on the descriptor.
is_open	Determine whether the descriptor is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native descriptor representation.
native_handle	Get the native descriptor representation.
native_non_blocking	Gets the non-blocking mode of the native descriptor implementation. Sets the non-blocking mode of the native descriptor implementation.
non_blocking	Gets the non-blocking mode of the descriptor. Sets the non-blocking mode of the descriptor.
operator=	Move-assign a basic_descriptor from another.
release	Release ownership of the native descriptor implementation.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_descriptor	Protected destructor to prevent deletion through this type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `posix::basic_descriptor` class template provides the ability to wrap a POSIX descriptor.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/posix/basic_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_descriptor::assign

Assign an existing native descriptor to the descriptor.

```
void assign(
    const native_handle_type & native_descriptor);
  » more...

boost::system::error_code assign(
    const native_handle_type & native_descriptor,
    boost::system::error_code & ec);
  » more...
```

posix::basic_descriptor::assign (1 of 2 overloads)

Assign an existing native descriptor to the descriptor.

```
void assign(
    const native_handle_type & native_descriptor);
```

posix::basic_descriptor::assign (2 of 2 overloads)

Assign an existing native descriptor to the descriptor.

```
boost::system::error_code assign(
    const native_handle_type & native_descriptor,
    boost::system::error_code & ec);
```

posix::basic_descriptor::basic_descriptor

Construct a `posix::basic_descriptor` without opening it.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
explicit basic_descriptor(
    boost::asio::io_service & io_service);
  » more...
```

Construct a `posix::basic_descriptor` on an existing native descriptor.

```
basic_descriptor(
    boost::asio::io_service & io_service,
    const native_handle_type & native_descriptor);
  » more...
```

Move-construct a `posix::basic_descriptor` from another.

```
basic_descriptor(
    basic_descriptor && other);
  » more...
```

posix::basic_descriptor::basic_descriptor (1 of 3 overloads)

Construct a `posix::basic_descriptor` without opening it.

```
basic_descriptor(
    boost::asio::io_service & io_service);
```

This constructor creates a descriptor without opening it.

Parameters

io_service	The `io_service` object that the descriptor will use to dispatch handlers for any asynchronous operations performed on the descriptor.

posix::basic_descriptor::basic_descriptor (2 of 3 overloads)

Construct a `posix::basic_descriptor` on an existing native descriptor.

```
basic_descriptor(
    boost::asio::io_service & io_service,
    const native_handle_type & native_descriptor);
```

This constructor creates a descriptor object to hold an existing native descriptor.

Parameters

io_service	The `io_service` object that the descriptor will use to dispatch handlers for any asynchronous operations performed on the descriptor.
native_descriptor	A native descriptor.

Exceptions

boost::system::system_error	Thrown on failure.

posix::basic_descriptor::basic_descriptor (3 of 3 overloads)

Move-construct a `posix::basic_descriptor` from another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
basic_descriptor(
    basic_descriptor && other);
```

This constructor moves a descriptor from one object to another.

Parameters

other The other `posix::basic_descriptor` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_descriptor(io_service&)` constructor.

posix::basic_descriptor::bytes_readable

Inherited from posix::descriptor_base.

IO control command to get the amount of data that can be read without blocking.

```
typedef implementation_defined bytes_readable;
```

Implements the FIONREAD IO control command.

Example

```
boost::asio::posix::stream_descriptor descriptor(io_service);
...
boost::asio::descriptor_base::bytes_readable command(true);
descriptor.io_control(command);
std::size_t bytes_readable = command.get();
```

Requirements

Header: `boost/asio/posix/basic_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_descriptor::cancel

Cancel all asynchronous operations associated with the descriptor.

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

posix::basic_descriptor::cancel (1 of 2 overloads)

Cancel all asynchronous operations associated with the descriptor.

```
void cancel();
```

This function causes all outstanding asynchronous read or write operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

posix::basic_descriptor::cancel (2 of 2 overloads)

Cancel all asynchronous operations associated with the descriptor.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function causes all outstanding asynchronous read or write operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Parameters

ec	Set to indicate what error occurred, if any.

posix::basic_descriptor::close

Close the descriptor.

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

posix::basic_descriptor::close (1 of 2 overloads)

Close the descriptor.

```
void close();
```

This function is used to close the descriptor. Any asynchronous read or write operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure. Note that, even if the function indicates an error, the underlying descriptor is closed.

posix::basic_descriptor::close (2 of 2 overloads)

Close the descriptor.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```

This function is used to close the descriptor. Any asynchronous read or write operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Parameters

ec	Set to indicate what error occurred, if any. Note that, even if the function indicates an error, the underlying descriptor is closed.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_descriptor::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

posix::basic_descriptor::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

posix::basic_descriptor::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

posix::basic_descriptor::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

posix::basic_descriptor::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

posix::basic_descriptor::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
service_type & get_service();
```

posix::basic_descriptor::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

posix::basic_descriptor::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

posix::basic_descriptor::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/posix/basic_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_descriptor::io_control

Perform an IO control command on the descriptor.

```
template<
    typename IoControlCommand>
void io_control(
    IoControlCommand & command);
  » more...

template<
    typename IoControlCommand>
boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
  » more...
```

posix::basic_descriptor::io_control (1 of 2 overloads)

Perform an IO control command on the descriptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename IoControlCommand>
void io_control(
    IoControlCommand & command);
```

This function is used to execute an IO control command on the descriptor.

Parameters

command The IO control command to be performed on the descriptor.

Exceptions

boost::system::system_error Thrown on failure.

Example

Getting the number of bytes ready to read:

```
boost::asio::posix::stream_descriptor descriptor(io_service);
...
boost::asio::posix::stream_descriptor::bytes_readable command;
descriptor.io_control(command);
std::size_t bytes_readable = command.get();
```

posix::basic_descriptor::io_control (2 of 2 overloads)

Perform an IO control command on the descriptor.

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
```

This function is used to execute an IO control command on the descriptor.

Parameters

command The IO control command to be performed on the descriptor.

ec Set to indicate what error occurred, if any.

Example

Getting the number of bytes ready to read:

```
boost::asio::posix::stream_descriptor descriptor(io_service);
...
boost::asio::posix::stream_descriptor::bytes_readable command;
boost::system::error_code ec;
descriptor.io_control(command, ec);
if (ec)
{
  // An error occurred.
}
std::size_t bytes_readable = command.get();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_descriptor::is_open

Determine whether the descriptor is open.

```
bool is_open() const;
```

posix::basic_descriptor::lowest_layer

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

posix::basic_descriptor::lowest_layer (1 of 2 overloads)

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

This function returns a reference to the lowest layer in a stack of layers. Since a `posix::basic_descriptor` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

posix::basic_descriptor::lowest_layer (2 of 2 overloads)

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

This function returns a const reference to the lowest layer in a stack of layers. Since a `posix::basic_descriptor` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A const reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

posix::basic_descriptor::lowest_layer_type

A `posix::basic_descriptor` is always the lowest layer.

```
typedef basic_descriptor< DescriptorService > lowest_layer_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
bytes_readable	IO control command to get the amount of data that can be read without blocking.
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_descriptor is always the lowest layer.
native_handle_type	The native representation of a descriptor.
native_type	(Deprecated: Use native_handle_type.) The native representation of a descriptor.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the descriptor.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native descriptor to the descriptor.
basic_descriptor	Construct a basic_descriptor without opening it. Construct a basic_descriptor on an existing native descriptor. Move-construct a basic_descriptor from another.
cancel	Cancel all asynchronous operations associated with the descriptor.
close	Close the descriptor.
get_io_service	Get the io_service associated with the object.
io_control	Perform an IO control command on the descriptor.
is_open	Determine whether the descriptor is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native descriptor representation.
native_handle	Get the native descriptor representation.
native_non_blocking	Gets the non-blocking mode of the native descriptor implementation. Sets the non-blocking mode of the native descriptor implementation.
non_blocking	Gets the non-blocking mode of the descriptor. Sets the non-blocking mode of the descriptor.
operator=	Move-assign a basic_descriptor from another.
release	Release ownership of the native descriptor implementation.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_descriptor	Protected destructor to prevent deletion through this type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `posix::basic_descriptor` class template provides the ability to wrap a POSIX descriptor.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/posix/basic_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_descriptor::native

(Deprecated: Use `native_handle()`.) Get the native descriptor representation.

```
native_type native();
```

This function may be used to obtain the underlying representation of the descriptor. This is intended to allow access to native descriptor functionality that is not otherwise provided.

posix::basic_descriptor::native_handle

Get the native descriptor representation.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying representation of the descriptor. This is intended to allow access to native descriptor functionality that is not otherwise provided.

posix::basic_descriptor::native_handle_type

The native representation of a descriptor.

```
typedef DescriptorService::native_handle_type native_handle_type;
```

Requirements

Header: `boost/asio/posix/basic_descriptor.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_descriptor::native_non_blocking

Gets the non-blocking mode of the native descriptor implementation.

```
bool native_non_blocking() const;
  » more...
```

Sets the non-blocking mode of the native descriptor implementation.

```
void native_non_blocking(
    bool mode);
  » more...

boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

posix::basic_descriptor::native_non_blocking (1 of 3 overloads)

Gets the non-blocking mode of the native descriptor implementation.

```
bool native_non_blocking() const;
```

This function is used to retrieve the non-blocking mode of the underlying native descriptor. This mode has no effect on the behaviour of the descriptor object's synchronous operations.

Return Value

`true` if the underlying descriptor is in non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Remarks

The current non-blocking mode is cached by the descriptor object. Consequently, the return value may be incorrect if the non-blocking mode was set directly on the native descriptor.

posix::basic_descriptor::native_non_blocking (2 of 3 overloads)

Sets the non-blocking mode of the native descriptor implementation.

```
void native_non_blocking(
    bool mode);
```

This function is used to modify the non-blocking mode of the underlying native descriptor. It has no effect on the behaviour of the descriptor object's synchronous operations.

Parameters

mode — If `true`, the underlying descriptor is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Exceptions

boost::system::system_error — Thrown on failure. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_descriptor::native_non_blocking (3 of 3 overloads)

Sets the non-blocking mode of the native descriptor implementation.

```
boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

This function is used to modify the non-blocking mode of the underlying native descriptor. It has no effect on the behaviour of the descriptor object's synchronous operations.

Parameters

mode — If `true`, the underlying descriptor is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

ec — Set to indicate what error occurred, if any. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

posix::basic_descriptor::native_type

(Deprecated: Use native_handle_type.) The native representation of a descriptor.

```
typedef DescriptorService::native_handle_type native_type;
```

Requirements

Header: `boost/asio/posix/basic_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_descriptor::non_blocking

Gets the non-blocking mode of the descriptor.

```
bool non_blocking() const;
  » more...
```

Sets the non-blocking mode of the descriptor.

```
void non_blocking(
    bool mode);
  » more...

boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

posix::basic_descriptor::non_blocking (1 of 3 overloads)

Gets the non-blocking mode of the descriptor.

```
bool non_blocking() const;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Return Value

`true` if the descriptor's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

posix::basic_descriptor::non_blocking (2 of 3 overloads)

Sets the non-blocking mode of the descriptor.

```
void non_blocking(
    bool mode);
```

Parameters

mode — If `true`, the descriptor's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Exceptions

boost::system::system_error — Thrown on failure.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

posix::basic_descriptor::non_blocking (3 of 3 overloads)

Sets the non-blocking mode of the descriptor.

```
boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

Parameters

mode — If `true`, the descriptor's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

ec — Set to indicate what error occurred, if any.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

posix::basic_descriptor::non_blocking_io

Inherited from posix::descriptor_base.

(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the descriptor.

```
typedef implementation_defined non_blocking_io;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Implements the FIONBIO IO control command.

Example

```
boost::asio::posix::stream_descriptor descriptor(io_service);
...
boost::asio::descriptor_base::non_blocking_io command(true);
descriptor.io_control(command);
```

Requirements

Header: `boost/asio/posix/basic_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_descriptor::operator=

Move-assign a `posix::basic_descriptor` from another.

```
basic_descriptor & operator=(
    basic_descriptor && other);
```

This assignment operator moves a descriptor from one object to another.

Parameters

other The other `posix::basic_descriptor` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_descriptor(io_service&)` constructor.

posix::basic_descriptor::release

Release ownership of the native descriptor implementation.

```
native_handle_type release();
```

This function may be used to obtain the underlying representation of the descriptor. After calling this function, `is_open()` returns false. The caller is responsible for closing the descriptor.

All outstanding asynchronous read or write operations will finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

posix::basic_descriptor::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_descriptor::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef DescriptorService service_type;
```

Requirements

Header: `boost/asio/posix/basic_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_descriptor::~basic_descriptor

Protected destructor to prevent deletion through this type.

```
~basic_descriptor();
```

posix::basic_stream_descriptor

Provides stream-oriented descriptor functionality.

```
template<
    typename StreamDescriptorService = stream_descriptor_service>
class basic_stream_descriptor :
  public posix::basic_descriptor< StreamDescriptorService >
```

Types

Name	Description
bytes_readable	IO control command to get the amount of data that can be read without blocking.
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_descriptor is always the lowest layer.
native_handle_type	The native representation of a descriptor.
native_type	(Deprecated: Use native_handle_type.) The native representation of a descriptor.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the descriptor.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native descriptor to the descriptor.
async_read_some	Start an asynchronous read.
async_write_some	Start an asynchronous write.
basic_stream_descriptor	Construct a basic_stream_descriptor without opening it. Construct a basic_stream_descriptor on an existing native descriptor. Move-construct a basic_stream_descriptor from another.
cancel	Cancel all asynchronous operations associated with the descriptor.
close	Close the descriptor.
get_io_service	Get the io_service associated with the object.
io_control	Perform an IO control command on the descriptor.
is_open	Determine whether the descriptor is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native descriptor representation.
native_handle	Get the native descriptor representation.
native_non_blocking	Gets the non-blocking mode of the native descriptor implementation. Sets the non-blocking mode of the native descriptor implementation.
non_blocking	Gets the non-blocking mode of the descriptor. Sets the non-blocking mode of the descriptor.
operator=	Move-assign a basic_stream_descriptor from another.
read_some	Read some data from the descriptor.
release	Release ownership of the native descriptor implementation.
write_some	Write some data to the descriptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `posix::basic_stream_descriptor` class template provides asynchronous and blocking stream-oriented descriptor functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/posix/basic_stream_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_stream_descriptor::assign

Assign an existing native descriptor to the descriptor.

```
void assign(
    const native_handle_type & native_descriptor);
  » more...

boost::system::error_code assign(
    const native_handle_type & native_descriptor,
    boost::system::error_code & ec);
  » more...
```

posix::basic_stream_descriptor::assign (1 of 2 overloads)

Inherited from posix::basic_descriptor.

Assign an existing native descriptor to the descriptor.

```
void assign(
    const native_handle_type & native_descriptor);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_stream_descriptor::assign (2 of 2 overloads)

Inherited from posix::basic_descriptor.

Assign an existing native descriptor to the descriptor.

```
boost::system::error_code assign(
    const native_handle_type & native_descriptor,
    boost::system::error_code & ec);
```

posix::basic_stream_descriptor::async_read_some

Start an asynchronous read.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_some(
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

This function is used to asynchronously read data from the stream descriptor. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be read. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes read.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The read operation may not read all of the requested number of bytes. Consider using the `async_read` function if you need to ensure that the requested amount of data is read before the asynchronous operation completes.

Example

To read into a single data buffer use the `buffer` function as follows:

```
descriptor.async_read_some(boost::asio::buffer(data, size), handler);
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_stream_descriptor::async_write_some

Start an asynchronous write.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_some(
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

This function is used to asynchronously write data to the stream descriptor. The function call always returns immediately.

Parameters

buffers — One or more data buffers to be written to the descriptor. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the write operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes written.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The write operation may not transmit all of the data to the peer. Consider using the `async_write` function if you need to ensure that all data is written before the asynchronous operation completes.

Example

To write a single data buffer use the `buffer` function as follows:

```
descriptor.async_write_some(boost::asio::buffer(data, size), handler);
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

posix::basic_stream_descriptor::basic_stream_descriptor

Construct a `posix::basic_stream_descriptor` without opening it.

```
explicit basic_stream_descriptor(
    boost::asio::io_service & io_service);
  » more...
```

Construct a `posix::basic_stream_descriptor` on an existing native descriptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
basic_stream_descriptor(
    boost::asio::io_service & io_service,
    const native_handle_type & native_descriptor);
  » more...
```

Move-construct a `posix::basic_stream_descriptor` from another.

```
basic_stream_descriptor(
    basic_stream_descriptor && other);
  » more...
```

posix::basic_stream_descriptor::basic_stream_descriptor (1 of 3 overloads)

Construct a `posix::basic_stream_descriptor` without opening it.

```
basic_stream_descriptor(
    boost::asio::io_service & io_service);
```

This constructor creates a stream descriptor without opening it. The descriptor needs to be opened and then connected or accepted before data can be sent or received on it.

Parameters

io_service	The `io_service` object that the stream descriptor will use to dispatch handlers for any asynchronous operations performed on the descriptor.

posix::basic_stream_descriptor::basic_stream_descriptor (2 of 3 overloads)

Construct a `posix::basic_stream_descriptor` on an existing native descriptor.

```
basic_stream_descriptor(
    boost::asio::io_service & io_service,
    const native_handle_type & native_descriptor);
```

This constructor creates a stream descriptor object to hold an existing native descriptor.

Parameters

io_service	The `io_service` object that the stream descriptor will use to dispatch handlers for any asynchronous operations performed on the descriptor.
native_descriptor	The new underlying descriptor implementation.

Exceptions

boost::system::system_error	Thrown on failure.

posix::basic_stream_descriptor::basic_stream_descriptor (3 of 3 overloads)

Move-construct a `posix::basic_stream_descriptor` from another.

```
basic_stream_descriptor(
    basic_stream_descriptor && other);
```

This constructor moves a stream descriptor from one object to another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

other The other `posix::basic_stream_descriptor` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_stream_descriptor(io_service&) constructor`.

posix::basic_stream_descriptor::bytes_readable

Inherited from posix::descriptor_base.

IO control command to get the amount of data that can be read without blocking.

```
typedef implementation_defined bytes_readable;
```

Implements the FIONREAD IO control command.

Example

```
boost::asio::posix::stream_descriptor descriptor(io_service);
...
boost::asio::descriptor_base::bytes_readable command(true);
descriptor.io_control(command);
std::size_t bytes_readable = command.get();
```

Requirements

Header: `boost/asio/posix/basic_stream_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_stream_descriptor::cancel

Cancel all asynchronous operations associated with the descriptor.

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

posix::basic_stream_descriptor::cancel (1 of 2 overloads)

Inherited from posix::basic_descriptor.

Cancel all asynchronous operations associated with the descriptor.

```
void cancel();
```

This function causes all outstanding asynchronous read or write operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error Thrown on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_stream_descriptor::cancel (2 of 2 overloads)

Inherited from posix::basic_descriptor.

Cancel all asynchronous operations associated with the descriptor.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function causes all outstanding asynchronous read or write operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Parameters

ec Set to indicate what error occurred, if any.

posix::basic_stream_descriptor::close

Close the descriptor.

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

posix::basic_stream_descriptor::close (1 of 2 overloads)

Inherited from posix::basic_descriptor.

Close the descriptor.

```
void close();
```

This function is used to close the descriptor. Any asynchronous read or write operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure. Note that, even if the function indicates an error, the underlying descriptor is closed.

posix::basic_stream_descriptor::close (2 of 2 overloads)

Inherited from posix::basic_descriptor.

Close the descriptor.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```

This function is used to close the descriptor. Any asynchronous read or write operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

ec Set to indicate what error occurred, if any. Note that, even if the function indicates an error, the underlying descriptor is closed.

posix::basic_stream_descriptor::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

posix::basic_stream_descriptor::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

posix::basic_stream_descriptor::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

posix::basic_stream_descriptor::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

posix::basic_stream_descriptor::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_stream_descriptor::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

posix::basic_stream_descriptor::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

posix::basic_stream_descriptor::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

posix::basic_stream_descriptor::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/posix/basic_stream_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_stream_descriptor::io_control

Perform an IO control command on the descriptor.

```
void io_control(
    IoControlCommand & command);
  » more...

boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
  » more...
```

posix::basic_stream_descriptor::io_control (1 of 2 overloads)

Inherited from posix::basic_descriptor.

Perform an IO control command on the descriptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename IoControlCommand>
void io_control(
    IoControlCommand & command);
```

This function is used to execute an IO control command on the descriptor.

Parameters

command The IO control command to be performed on the descriptor.

Exceptions

boost::system::system_error Thrown on failure.

Example

Getting the number of bytes ready to read:

```
boost::asio::posix::stream_descriptor descriptor(io_service);
...
boost::asio::posix::stream_descriptor::bytes_readable command;
descriptor.io_control(command);
std::size_t bytes_readable = command.get();
```

posix::basic_stream_descriptor::io_control (2 of 2 overloads)

Inherited from posix::basic_descriptor.

Perform an IO control command on the descriptor.

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    IoControlCommand & command,
    boost::system::error_code & ec);
```

This function is used to execute an IO control command on the descriptor.

Parameters

command The IO control command to be performed on the descriptor.

ec Set to indicate what error occurred, if any.

Example

Getting the number of bytes ready to read:

```
boost::asio::posix::stream_descriptor descriptor(io_service);
...
boost::asio::posix::stream_descriptor::bytes_readable command;
boost::system::error_code ec;
descriptor.io_control(command, ec);
if (ec)
{
  // An error occurred.
}
std::size_t bytes_readable = command.get();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_stream_descriptor::is_open

Inherited from posix::basic_descriptor.

Determine whether the descriptor is open.

```
bool is_open() const;
```

posix::basic_stream_descriptor::lowest_layer

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

posix::basic_stream_descriptor::lowest_layer (1 of 2 overloads)

Inherited from posix::basic_descriptor.

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

This function returns a reference to the lowest layer in a stack of layers. Since a `posix::basic_descriptor` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

posix::basic_stream_descriptor::lowest_layer (2 of 2 overloads)

Inherited from posix::basic_descriptor.

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

This function returns a const reference to the lowest layer in a stack of layers. Since a `posix::basic_descriptor` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A const reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

posix::basic_stream_descriptor::lowest_layer_type

Inherited from posix::basic_descriptor.

A `posix::basic_descriptor` is always the lowest layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef basic_descriptor< StreamDescriptorService > lowest_layer_type;
```

Types

Name	Description
bytes_readable	IO control command to get the amount of data that can be read without blocking.
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_descriptor is always the lowest layer.
native_handle_type	The native representation of a descriptor.
native_type	(Deprecated: Use native_handle_type.) The native representation of a descriptor.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the descriptor.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native descriptor to the descriptor.
basic_descriptor	Construct a basic_descriptor without opening it. Construct a basic_descriptor on an existing native descriptor. Move-construct a basic_descriptor from another.
cancel	Cancel all asynchronous operations associated with the descriptor.
close	Close the descriptor.
get_io_service	Get the io_service associated with the object.
io_control	Perform an IO control command on the descriptor.
is_open	Determine whether the descriptor is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native descriptor representation.
native_handle	Get the native descriptor representation.
native_non_blocking	Gets the non-blocking mode of the native descriptor implementation. Sets the non-blocking mode of the native descriptor implementation.
non_blocking	Gets the non-blocking mode of the descriptor. Sets the non-blocking mode of the descriptor.
operator=	Move-assign a basic_descriptor from another.
release	Release ownership of the native descriptor implementation.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_descriptor	Protected destructor to prevent deletion through this type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `posix::basic_descriptor` class template provides the ability to wrap a POSIX descriptor.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/posix/basic_stream_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_stream_descriptor::native

Inherited from posix::basic_descriptor.

(Deprecated: Use `native_handle()`.) Get the native descriptor representation.

```
native_type native();
```

This function may be used to obtain the underlying representation of the descriptor. This is intended to allow access to native descriptor functionality that is not otherwise provided.

posix::basic_stream_descriptor::native_handle

Inherited from posix::basic_descriptor.

Get the native descriptor representation.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying representation of the descriptor. This is intended to allow access to native descriptor functionality that is not otherwise provided.

posix::basic_stream_descriptor::native_handle_type

The native representation of a descriptor.

```
typedef StreamDescriptorService::native_handle_type native_handle_type;
```

Requirements

Header: `boost/asio/posix/basic_stream_descriptor.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_stream_descriptor::native_non_blocking

Gets the non-blocking mode of the native descriptor implementation.

```
bool native_non_blocking() const;
  » more...
```

Sets the non-blocking mode of the native descriptor implementation.

```
void native_non_blocking(
    bool mode);
  » more...

boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```

posix::basic_stream_descriptor::native_non_blocking (1 of 3 overloads)

Inherited from posix::basic_descriptor.

Gets the non-blocking mode of the native descriptor implementation.

```
bool native_non_blocking() const;
```

This function is used to retrieve the non-blocking mode of the underlying native descriptor. This mode has no effect on the behaviour of the descriptor object's synchronous operations.

Return Value

`true` if the underlying descriptor is in non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

Remarks

The current non-blocking mode is cached by the descriptor object. Consequently, the return value may be incorrect if the non-blocking mode was set directly on the native descriptor.

posix::basic_stream_descriptor::native_non_blocking (2 of 3 overloads)

Inherited from posix::basic_descriptor.

Sets the non-blocking mode of the native descriptor implementation.

```
void native_non_blocking(
    bool mode);
```

This function is used to modify the non-blocking mode of the underlying native descriptor. It has no effect on the behaviour of the descriptor object's synchronous operations.

Parameters

mode — If `true`, the underlying descriptor is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

posix::basic_stream_descriptor::native_non_blocking (3 of 3 overloads)

Inherited from posix::basic_descriptor.

Sets the non-blocking mode of the native descriptor implementation.

```
boost::system::error_code native_non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

This function is used to modify the non-blocking mode of the underlying native descriptor. It has no effect on the behaviour of the descriptor object's synchronous operations.

Parameters

mode	If `true`, the underlying descriptor is put into non-blocking mode and direct system calls may fail with `boost::asio::error::would_block` (or the equivalent system error).
ec	Set to indicate what error occurred, if any. If the `mode` is `false`, but the current value of `non_blocking()` is `true`, this function fails with `boost::asio::error::invalid_argument`, as the combination does not make sense.

posix::basic_stream_descriptor::native_type

(Deprecated: Use native_handle_type.) The native representation of a descriptor.

```
typedef StreamDescriptorService::native_handle_type native_type;
```

Requirements

Header: `boost/asio/posix/basic_stream_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_stream_descriptor::non_blocking

Gets the non-blocking mode of the descriptor.

```
bool non_blocking() const;
  » more...
```

Sets the non-blocking mode of the descriptor.

```
void non_blocking(
    bool mode);
  » more...

boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_stream_descriptor::non_blocking (1 of 3 overloads)

Inherited from posix::basic_descriptor.

Gets the non-blocking mode of the descriptor.

```
bool non_blocking() const;
```

Return Value

`true` if the descriptor's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

posix::basic_stream_descriptor::non_blocking (2 of 3 overloads)

Inherited from posix::basic_descriptor.

Sets the non-blocking mode of the descriptor.

```
void non_blocking(
    bool mode);
```

Parameters

mode — If `true`, the descriptor's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

Exceptions

boost::system::system_error — Thrown on failure.

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

posix::basic_stream_descriptor::non_blocking (3 of 3 overloads)

Inherited from posix::basic_descriptor.

Sets the non-blocking mode of the descriptor.

```
boost::system::error_code non_blocking(
    bool mode,
    boost::system::error_code & ec);
```

Parameters

mode — If `true`, the descriptor's synchronous operations will fail with `boost::asio::error::would_block` if they are unable to perform the requested operation immediately. If `false`, synchronous operations will block until complete.

ec — Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

The non-blocking mode has no effect on the behaviour of asynchronous operations. Asynchronous operations will never fail with the error `boost::asio::error::would_block`.

posix::basic_stream_descriptor::non_blocking_io

Inherited from posix::descriptor_base.

(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the descriptor.

```
typedef implementation_defined non_blocking_io;
```

Implements the FIONBIO IO control command.

Example

```
boost::asio::posix::stream_descriptor descriptor(io_service);
...
boost::asio::descriptor_base::non_blocking_io command(true);
descriptor.io_control(command);
```

Requirements

Header: `boost/asio/posix/basic_stream_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_stream_descriptor::operator=

Move-assign a `posix::basic_stream_descriptor` from another.

```
basic_stream_descriptor & operator=(
    basic_stream_descriptor && other);
```

This assignment operator moves a stream descriptor from one object to another.

Parameters

other The other `posix::basic_stream_descriptor` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_stream_descriptor(io_service&) constructor`.

posix::basic_stream_descriptor::read_some

Read some data from the descriptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
  » more...

template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

posix::basic_stream_descriptor::read_some (1 of 2 overloads)

Read some data from the descriptor.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
```

This function is used to read data from the stream descriptor. The function call will block until one or more bytes of data has been read successfully, or until an error occurs.

Parameters

buffers One or more buffers into which the data will be read.

Return Value

The number of bytes read.

Exceptions

boost::system::system_error Thrown on failure. An error code of `boost::asio::error::eof` indicates that the connection was closed by the peer.

Remarks

The read_some operation may not read all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

Example

To read into a single data buffer use the `buffer` function as follows:

```
descriptor.read_some(boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

posix::basic_stream_descriptor::read_some (2 of 2 overloads)

Read some data from the descriptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to read data from the stream descriptor. The function call will block until one or more bytes of data has been read successfully, or until an error occurs.

Parameters

buffers One or more buffers into which the data will be read.

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes read. Returns 0 if an error occurred.

Remarks

The read_some operation may not read all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

posix::basic_stream_descriptor::release

Inherited from posix::basic_descriptor.

Release ownership of the native descriptor implementation.

```
native_handle_type release();
```

This function may be used to obtain the underlying representation of the descriptor. After calling this function, `is_open()` returns false. The caller is responsible for closing the descriptor.

All outstanding asynchronous read or write operations will finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

posix::basic_stream_descriptor::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

posix::basic_stream_descriptor::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef StreamDescriptorService service_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/posix/basic_stream_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::basic_stream_descriptor::write_some

Write some data to the descriptor.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
  » more...

template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

posix::basic_stream_descriptor::write_some (1 of 2 overloads)

Write some data to the descriptor.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
```

This function is used to write data to the stream descriptor. The function call will block until one or more bytes of the data has been written successfully, or until an error occurs.

Parameters

buffers	One or more data buffers to be written to the descriptor.

Return Value

The number of bytes written.

Exceptions

boost::system::system_error	Thrown on failure. An error code of `boost::asio::error::eof` indicates that the connection was closed by the peer.

Remarks

The write_some operation may not transmit all of the data to the peer. Consider using the `write` function if you need to ensure that all data is written before the blocking operation completes.

Example

To write a single data buffer use the `buffer` function as follows:

```
descriptor.write_some(boost::asio::buffer(data, size));
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

posix::basic_stream_descriptor::write_some (2 of 2 overloads)

Write some data to the descriptor.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to write data to the stream descriptor. The function call will block until one or more bytes of the data has been written successfully, or until an error occurs.

Parameters

buffers One or more data buffers to be written to the descriptor.

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes written. Returns 0 if an error occurred.

Remarks

The write_some operation may not transmit all of the data to the peer. Consider using the `write` function if you need to ensure that all data is written before the blocking operation completes.

posix::descriptor_base

The `posix::descriptor_base` class is used as a base for the `posix::basic_stream_descriptor` class template so that we have a common place to define the associated IO control commands.

```
class descriptor_base
```

Types

Name	Description
bytes_readable	IO control command to get the amount of data that can be read without blocking.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the descriptor.

Protected Member Functions

Name	Description
~descriptor_base	Protected destructor to prevent deletion through this type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/posix/descriptor_base.hpp`

Convenience header: `boost/asio.hpp`

posix::descriptor_base::bytes_readable

IO control command to get the amount of data that can be read without blocking.

```
typedef implementation_defined bytes_readable;
```

Implements the FIONREAD IO control command.

Example

```
boost::asio::posix::stream_descriptor descriptor(io_service);
...
boost::asio::descriptor_base::bytes_readable command(true);
descriptor.io_control(command);
std::size_t bytes_readable = command.get();
```

Requirements

Header: `boost/asio/posix/descriptor_base.hpp`

Convenience header: `boost/asio.hpp`

posix::descriptor_base::non_blocking_io

(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the descriptor.

```
typedef implementation_defined non_blocking_io;
```

Implements the FIONBIO IO control command.

Example

```
boost::asio::posix::stream_descriptor descriptor(io_service);
...
boost::asio::descriptor_base::non_blocking_io command(true);
descriptor.io_control(command);
```

Requirements

Header: `boost/asio/posix/descriptor_base.hpp`

Convenience header: `boost/asio.hpp`

posix::descriptor_base::~descriptor_base

Protected destructor to prevent deletion through this type.

```
~descriptor_base();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::stream_descriptor

Typedef for the typical usage of a stream-oriented descriptor.

```
typedef basic_stream_descriptor stream_descriptor;
```

Types

Name	Description
bytes_readable	IO control command to get the amount of data that can be read without blocking.
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_descriptor is always the lowest layer.
native_handle_type	The native representation of a descriptor.
native_type	(Deprecated: Use native_handle_type.) The native representation of a descriptor.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the descriptor.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native descriptor to the descriptor.
async_read_some	Start an asynchronous read.
async_write_some	Start an asynchronous write.
basic_stream_descriptor	Construct a basic_stream_descriptor without opening it. Construct a basic_stream_descriptor on an existing native descriptor. Move-construct a basic_stream_descriptor from another.
cancel	Cancel all asynchronous operations associated with the descriptor.
close	Close the descriptor.
get_io_service	Get the io_service associated with the object.
io_control	Perform an IO control command on the descriptor.
is_open	Determine whether the descriptor is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native descriptor representation.
native_handle	Get the native descriptor representation.
native_non_blocking	Gets the non-blocking mode of the native descriptor implementation. Sets the non-blocking mode of the native descriptor implementation.
non_blocking	Gets the non-blocking mode of the descriptor. Sets the non-blocking mode of the descriptor.
operator=	Move-assign a basic_stream_descriptor from another.
read_some	Read some data from the descriptor.
release	Release ownership of the native descriptor implementation.
write_some	Write some data to the descriptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `posix::basic_stream_descriptor` class template provides asynchronous and blocking stream-oriented descriptor functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/posix/stream_descriptor.hpp`

Convenience header: `boost/asio.hpp`

posix::stream_descriptor_service

Default service implementation for a stream descriptor.

```
class stream_descriptor_service :
  public io_service::service
```

Types

Name	Description
implementation_type	The type of a stream descriptor implementation.
native_handle_type	The native descriptor type.
native_type	(Deprecated: Use native_handle_type.) The native descriptor type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native descriptor to a stream descriptor.
async_read_some	Start an asynchronous read.
async_write_some	Start an asynchronous write.
cancel	Cancel all asynchronous operations associated with the descriptor.
close	Close a stream descriptor implementation.
construct	Construct a new stream descriptor implementation.
destroy	Destroy a stream descriptor implementation.
get_io_service	Get the io_service object that owns the service.
io_control	Perform an IO control command on the descriptor.
is_open	Determine whether the descriptor is open.
move_assign	Move-assign from another stream descriptor implementation.
move_construct	Move-construct a new stream descriptor implementation.
native	(Deprecated: Use native_handle().) Get the native descriptor implementation.
native_handle	Get the native descriptor implementation.
native_non_blocking	Gets the non-blocking mode of the native descriptor implementation. Sets the non-blocking mode of the native descriptor implementation.
non_blocking	Gets the non-blocking mode of the descriptor. Sets the non-blocking mode of the descriptor.
read_some	Read some data from the stream.
release	Release ownership of the native descriptor implementation.
stream_descriptor_service	Construct a new stream descriptor service for the specified io_service.
write_some	Write the given data to the stream.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Data Members

Name	Description
id	The unique service identifier.

Requirements

Header: `boost/asio/posix/stream_descriptor_service.hpp`

Convenience header: `boost/asio.hpp`

posix::stream_descriptor_service::assign

Assign an existing native descriptor to a stream descriptor.

```
boost::system::error_code assign(
    implementation_type & impl,
    const native_handle_type & native_descriptor,
    boost::system::error_code & ec);
```

posix::stream_descriptor_service::async_read_some

Start an asynchronous read.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_some(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

posix::stream_descriptor_service::async_write_some

Start an asynchronous write.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_some(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

posix::stream_descriptor_service::cancel

Cancel all asynchronous operations associated with the descriptor.

```
boost::system::error_code cancel(
    implementation_type & impl,
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::stream_descriptor_service::close

Close a stream descriptor implementation.

```
boost::system::error_code close(
    implementation_type & impl,
    boost::system::error_code & ec);
```

posix::stream_descriptor_service::construct

Construct a new stream descriptor implementation.

```
void construct(
    implementation_type & impl);
```

posix::stream_descriptor_service::destroy

Destroy a stream descriptor implementation.

```
void destroy(
    implementation_type & impl);
```

posix::stream_descriptor_service::get_io_service

Inherited from io_service.

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

posix::stream_descriptor_service::id

The unique service identifier.

```
static boost::asio::io_service::id id;
```

posix::stream_descriptor_service::implementation_type

The type of a stream descriptor implementation.

```
typedef implementation_defined implementation_type;
```

Requirements

Header: `boost/asio/posix/stream_descriptor_service.hpp`

Convenience header: `boost/asio.hpp`

posix::stream_descriptor_service::io_control

Perform an IO control command on the descriptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    implementation_type & impl,
    IoControlCommand & command,
    boost::system::error_code & ec);
```

posix::stream_descriptor_service::is_open

Determine whether the descriptor is open.

```
bool is_open(
    const implementation_type & impl) const;
```

posix::stream_descriptor_service::move_assign

Move-assign from another stream descriptor implementation.

```
void move_assign(
    implementation_type & impl,
    stream_descriptor_service & other_service,
    implementation_type & other_impl);
```

posix::stream_descriptor_service::move_construct

Move-construct a new stream descriptor implementation.

```
void move_construct(
    implementation_type & impl,
    implementation_type & other_impl);
```

posix::stream_descriptor_service::native

(Deprecated: Use `native_handle()`.) Get the native descriptor implementation.

```
native_type native(
    implementation_type & impl);
```

posix::stream_descriptor_service::native_handle

Get the native descriptor implementation.

```
native_handle_type native_handle(
    implementation_type & impl);
```

posix::stream_descriptor_service::native_handle_type

The native descriptor type.

```
typedef implementation_defined native_handle_type;
```

Requirements

Header: `boost/asio/posix/stream_descriptor_service.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio.hpp`

posix::stream_descriptor_service::native_non_blocking

Gets the non-blocking mode of the native descriptor implementation.

```
bool native_non_blocking(
    const implementation_type & impl) const;
  » more...
```

Sets the non-blocking mode of the native descriptor implementation.

```
boost::system::error_code native_non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
  » more...
```

posix::stream_descriptor_service::native_non_blocking (1 of 2 overloads)

Gets the non-blocking mode of the native descriptor implementation.

```
bool native_non_blocking(
    const implementation_type & impl) const;
```

posix::stream_descriptor_service::native_non_blocking (2 of 2 overloads)

Sets the non-blocking mode of the native descriptor implementation.

```
boost::system::error_code native_non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
```

posix::stream_descriptor_service::native_type

(Deprecated: Use native_handle_type.) The native descriptor type.

```
typedef implementation_defined native_type;
```

Requirements

Header: `boost/asio/posix/stream_descriptor_service.hpp`

Convenience header: `boost/asio.hpp`

posix::stream_descriptor_service::non_blocking

Gets the non-blocking mode of the descriptor.

```
bool non_blocking(
    const implementation_type & impl) const;
  » more...
```

Sets the non-blocking mode of the descriptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
  » more...
```

posix::stream_descriptor_service::non_blocking (1 of 2 overloads)

Gets the non-blocking mode of the descriptor.

```
bool non_blocking(
    const implementation_type & impl) const;
```

posix::stream_descriptor_service::non_blocking (2 of 2 overloads)

Sets the non-blocking mode of the descriptor.

```
boost::system::error_code non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
```

posix::stream_descriptor_service::read_some

Read some data from the stream.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

posix::stream_descriptor_service::release

Release ownership of the native descriptor implementation.

```
native_handle_type release(
    implementation_type & impl);
```

posix::stream_descriptor_service::stream_descriptor_service

Construct a new stream descriptor service for the specified `io_service`.

```
stream_descriptor_service(
    boost::asio::io_service & io_service);
```

posix::stream_descriptor_service::write_some

Write the given data to the stream.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

raw_socket_service

Default service implementation for a raw socket.

```
template<
    typename Protocol>
class raw_socket_service :
  public io_service::service
```

Types

Name	Description
endpoint_type	The endpoint type.
implementation_type	The type of a raw socket.
native_handle_type	The native socket type.
native_type	(Deprecated: Use native_handle_type.) The native socket type.
protocol_type	The protocol type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to a raw socket.
async_connect	Start an asynchronous connect.
async_receive	Start an asynchronous receive.
async_receive_from	Start an asynchronous receive that will get the endpoint of the sender.
async_send	Start an asynchronous send.
async_send_to	Start an asynchronous send.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
bind	
cancel	Cancel all asynchronous operations associated with the socket.
close	Close a raw socket implementation.
connect	Connect the raw socket to the specified endpoint.
construct	Construct a new raw socket implementation.
destroy	Destroy a raw socket implementation.
get_io_service	Get the io_service object that owns the service.
get_option	Get a socket option.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint.
move_assign	Move-assign from another raw socket implementation.
move_construct	Move-construct a new raw socket implementation.
native	(Deprecated: Use native_handle().) Get the native socket implementation.
native_handle	Get the native socket implementation.
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
open	
raw_socket_service	Construct a new raw socket service for the specified io_service.
receive	Receive some data from the peer.
receive_from	Receive raw data with the endpoint of the sender.
remote_endpoint	Get the remote endpoint.
send	Send the given data to the peer.
send_to	Send raw data to the specified endpoint.
set_option	Set a socket option.
shutdown	Disable sends or receives on the socket.

Data Members

Name	Description
id	The unique service identifier.

Requirements

Header: `boost/asio/raw_socket_service.hpp`

Convenience header: `boost/asio.hpp`

raw_socket_service::assign

Assign an existing native socket to a raw socket.

```
boost::system::error_code assign(
    implementation_type & impl,
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
```

raw_socket_service::async_connect

Start an asynchronous connect.

```
template<
    typename ConnectHandler>
void async_connect(
    implementation_type & impl,
    const endpoint_type & peer_endpoint,
    ConnectHandler handler);
```

raw_socket_service::async_receive

Start an asynchronous receive.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    ReadHandler handler);
```

raw_socket_service::async_receive_from

Start an asynchronous receive that will get the endpoint of the sender.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive_from(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags,
    ReadHandler handler);
```

raw_socket_service::async_send

Start an asynchronous send.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    WriteHandler handler);
```

raw_socket_service::async_send_to

Start an asynchronous send.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send_to(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags,
    WriteHandler handler);
```

raw_socket_service::at_mark

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

raw_socket_service::available

Determine the number of bytes available for reading.

```
std::size_t available(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

raw_socket_service::bind

```
boost::system::error_code bind(
    implementation_type & impl,
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
```

raw_socket_service::cancel

Cancel all asynchronous operations associated with the socket.

```
boost::system::error_code cancel(
    implementation_type & impl,
    boost::system::error_code & ec);
```

raw_socket_service::close

Close a raw socket implementation.

```
boost::system::error_code close(
    implementation_type & impl,
    boost::system::error_code & ec);
```

raw_socket_service::connect

Connect the raw socket to the specified endpoint.

```
boost::system::error_code connect(
    implementation_type & impl,
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
```

raw_socket_service::construct

Construct a new raw socket implementation.

```
void construct(
    implementation_type & impl);
```

raw_socket_service::destroy

Destroy a raw socket implementation.

```
void destroy(
    implementation_type & impl);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

raw_socket_service::endpoint_type

The endpoint type.

```
typedef Protocol::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/raw_socket_service.hpp`

Convenience header: `boost/asio.hpp`

raw_socket_service::get_io_service

Inherited from io_service.

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

raw_socket_service::get_option

Get a socket option.

```
template<
    typename GettableSocketOption>
boost::system::error_code get_option(
    const implementation_type & impl,
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
```

raw_socket_service::id

The unique service identifier.

```
static boost::asio::io_service::id id;
```

raw_socket_service::implementation_type

The type of a raw socket.

```
typedef implementation_defined implementation_type;
```

Requirements

Header: `boost/asio/raw_socket_service.hpp`

Convenience header: `boost/asio.hpp`

raw_socket_service::io_control

Perform an IO control command on the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    implementation_type & impl,
    IoControlCommand & command,
    boost::system::error_code & ec);
```

raw_socket_service::is_open

Determine whether the socket is open.

```
bool is_open(
    const implementation_type & impl) const;
```

raw_socket_service::local_endpoint

Get the local endpoint.

```
endpoint_type local_endpoint(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

raw_socket_service::move_assign

Move-assign from another raw socket implementation.

```
void move_assign(
    implementation_type & impl,
    raw_socket_service & other_service,
    implementation_type & other_impl);
```

raw_socket_service::move_construct

Move-construct a new raw socket implementation.

```
void move_construct(
    implementation_type & impl,
    implementation_type & other_impl);
```

raw_socket_service::native

(Deprecated: Use `native_handle()`.) Get the native socket implementation.

```
native_type native(
    implementation_type & impl);
```

raw_socket_service::native_handle

Get the native socket implementation.

```
native_handle_type native_handle(
    implementation_type & impl);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

raw_socket_service::native_handle_type

The native socket type.

```
typedef implementation_defined native_handle_type;
```

Requirements

Header: `boost/asio/raw_socket_service.hpp`

Convenience header: `boost/asio.hpp`

raw_socket_service::native_non_blocking

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking(
    const implementation_type & impl) const;
  » more...
```

Sets the non-blocking mode of the native socket implementation.

```
boost::system::error_code native_non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
  » more...
```

raw_socket_service::native_non_blocking (1 of 2 overloads)

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking(
    const implementation_type & impl) const;
```

raw_socket_service::native_non_blocking (2 of 2 overloads)

Sets the non-blocking mode of the native socket implementation.

```
boost::system::error_code native_non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
```

raw_socket_service::native_type

(Deprecated: Use native_handle_type.) The native socket type.

```
typedef implementation_defined native_type;
```

Requirements

Header: `boost/asio/raw_socket_service.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

raw_socket_service::non_blocking

Gets the non-blocking mode of the socket.

```
bool non_blocking(
    const implementation_type & impl) const;
  » more...
```

Sets the non-blocking mode of the socket.

```
boost::system::error_code non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
  » more...
```

raw_socket_service::non_blocking (1 of 2 overloads)

Gets the non-blocking mode of the socket.

```
bool non_blocking(
    const implementation_type & impl) const;
```

raw_socket_service::non_blocking (2 of 2 overloads)

Sets the non-blocking mode of the socket.

```
boost::system::error_code non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
```

raw_socket_service::open

```
boost::system::error_code open(
    implementation_type & impl,
    const protocol_type & protocol,
    boost::system::error_code & ec);
```

raw_socket_service::protocol_type

The protocol type.

```
typedef Protocol protocol_type;
```

Requirements

Header: `boost/asio/raw_socket_service.hpp`

Convenience header: `boost/asio.hpp`

raw_socket_service::raw_socket_service

Construct a new raw socket service for the specified `io_service`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
raw_socket_service(
    boost::asio::io_service & io_service);
```

raw_socket_service::receive

Receive some data from the peer.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

raw_socket_service::receive_from

Receive raw data with the endpoint of the sender.

```
template<
    typename MutableBufferSequence>
std::size_t receive_from(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    endpoint_type & sender_endpoint,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

raw_socket_service::remote_endpoint

Get the remote endpoint.

```
endpoint_type remote_endpoint(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

raw_socket_service::send

Send the given data to the peer.

```
template<
    typename ConstBufferSequence>
std::size_t send(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

raw_socket_service::send_to

Send raw data to the specified endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t send_to(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    const endpoint_type & destination,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

raw_socket_service::set_option

Set a socket option.

```
template<
    typename SettableSocketOption>
boost::system::error_code set_option(
    implementation_type & impl,
    const SettableSocketOption & option,
    boost::system::error_code & ec);
```

raw_socket_service::shutdown

Disable sends or receives on the socket.

```
boost::system::error_code shutdown(
    implementation_type & impl,
    socket_base::shutdown_type what,
    boost::system::error_code & ec);
```

read

Attempt to read a certain amount of data from a stream before returning.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename SyncReadStream,
    typename MutableBufferSequence>
std::size_t read(
    SyncReadStream & s,
    const MutableBufferSequence & buffers);
  » more...

template<
    typename SyncReadStream,
    typename MutableBufferSequence>
std::size_t read(
    SyncReadStream & s,
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...

template<
    typename SyncReadStream,
    typename MutableBufferSequence,
    typename CompletionCondition>
std::size_t read(
    SyncReadStream & s,
    const MutableBufferSequence & buffers,
    CompletionCondition completion_condition);
  » more...

template<
    typename SyncReadStream,
    typename MutableBufferSequence,
    typename CompletionCondition>
std::size_t read(
    SyncReadStream & s,
    const MutableBufferSequence & buffers,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
  » more...

template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read(
    SyncReadStream & s,
    basic_streambuf< Allocator > & b);
  » more...

template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read(
    SyncReadStream & s,
    basic_streambuf< Allocator > & b,
    boost::system::error_code & ec);
  » more...

template<
    typename SyncReadStream,
    typename Allocator,
    typename CompletionCondition>
std::size_t read(
    SyncReadStream & s,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
  » more...

template<
    typename SyncReadStream,
    typename Allocator,
    typename CompletionCondition>
std::size_t read(
    SyncReadStream & s,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
  » more...
```

Requirements

Header: `boost/asio/read.hpp`

Convenience header: `boost/asio.hpp`

read (1 of 8 overloads)

Attempt to read a certain amount of data from a stream before returning.

```
template<
    typename SyncReadStream,
    typename MutableBufferSequence>
std::size_t read(
    SyncReadStream & s,
    const MutableBufferSequence & buffers);
```

This function is used to read a certain number of bytes of data from a stream. The call will block until one of the following conditions is true:

- The supplied buffers are full. That is, the bytes transferred is equal to the sum of the buffer sizes.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's read_some function.

Parameters

s	The stream from which the data is to be read. The type must support the SyncReadStream concept.
buffers	One or more buffers into which the data will be read. The sum of the buffer sizes indicates the maximum number of bytes to read from the stream.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error	Thrown on failure.

Example

To read into a single data buffer use the `buffer` function as follows:

```
boost::asio::read(s, boost::asio::buffer(data, size));
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

See the buffer documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

Remarks

This overload is equivalent to calling:

```
boost::asio::read(
    s, buffers,
    boost::asio::transfer_all());
```

read (2 of 8 overloads)

Attempt to read a certain amount of data from a stream before returning.

```
template<
    typename SyncReadStream,
    typename MutableBufferSequence>
std::size_t read(
    SyncReadStream & s,
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to read a certain number of bytes of data from a stream. The call will block until one of the following conditions is true:

- The supplied buffers are full. That is, the bytes transferred is equal to the sum of the buffer sizes.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's read_some function.

Parameters

s	The stream from which the data is to be read. The type must support the SyncReadStream concept.
buffers	One or more buffers into which the data will be read. The sum of the buffer sizes indicates the maximum number of bytes to read from the stream.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes transferred.

Example

To read into a single data buffer use the buffer function as follows:

```
boost::asio::read(s, boost::asio::buffer(data, size), ec);
```

See the buffer documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

Remarks

This overload is equivalent to calling:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::read(
    s, buffers,
    boost::asio::transfer_all(), ec);
```

read (3 of 8 overloads)

Attempt to read a certain amount of data from a stream before returning.

```
template<
    typename SyncReadStream,
    typename MutableBufferSequence,
    typename CompletionCondition>
std::size_t read(
    SyncReadStream & s,
    const MutableBufferSequence & buffers,
    CompletionCondition completion_condition);
```

This function is used to read a certain number of bytes of data from a stream. The call will block until one of the following conditions is true:

- The supplied buffers are full. That is, the bytes transferred is equal to the sum of the buffer sizes.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the stream's read_some function.

Parameters

s — The stream from which the data is to be read. The type must support the SyncReadStream concept.

buffers — One or more buffers into which the data will be read. The sum of the buffer sizes indicates the maximum number of bytes to read from the stream.

completion_condition — The function object to be called to determine whether the read operation is complete. The signature of the function object must be:

```
std::size_t completion_condition(
  // Result of latest read_some operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the read operation is complete. A non-zero return value indicates the maximum number of bytes to be read on the next call to the stream's read_some function.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error — Thrown on failure.

Example

To read into a single data buffer use the `buffer` function as follows:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::read(s, boost::asio::buffer(data, size),
    boost::asio::transfer_at_least(32));
```

See the buffer documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

read (4 of 8 overloads)

Attempt to read a certain amount of data from a stream before returning.

```
template<
    typename SyncReadStream,
    typename MutableBufferSequence,
    typename CompletionCondition>
std::size_t read(
    SyncReadStream & s,
    const MutableBufferSequence & buffers,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
```

This function is used to read a certain number of bytes of data from a stream. The call will block until one of the following conditions is true:

- The supplied buffers are full. That is, the bytes transferred is equal to the sum of the buffer sizes.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the stream's read_some function.

Parameters

s — The stream from which the data is to be read. The type must support the SyncReadStream concept.

buffers — One or more buffers into which the data will be read. The sum of the buffer sizes indicates the maximum number of bytes to read from the stream.

completion_condition — The function object to be called to determine whether the read operation is complete. The signature of the function object must be:

```
std::size_t completion_condition(
  // Result of latest read_some operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the read operation is complete. A non-zero return value indicates the maximum number of bytes to be read on the next call to the stream's read_some function.

ec — Set to indicate what error occurred, if any.

Return Value

The number of bytes read. If an error occurs, returns the total number of bytes successfully transferred prior to the error.

read (5 of 8 overloads)

Attempt to read a certain amount of data from a stream before returning.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read(
    SyncReadStream & s,
    basic_streambuf< Allocator > & b);
```

This function is used to read a certain number of bytes of data from a stream. The call will block until one of the following conditions is true:

- The supplied buffer is full (that is, it has reached maximum size).
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's read_some function.

Parameters

s The stream from which the data is to be read. The type must support the SyncReadStream concept.

b The `basic_streambuf` object into which the data will be read.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

This overload is equivalent to calling:

```
boost::asio::read(
    s, b,
    boost::asio::transfer_all());
```

read (6 of 8 overloads)

Attempt to read a certain amount of data from a stream before returning.

```
template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read(
    SyncReadStream & s,
    basic_streambuf< Allocator > & b,
    boost::system::error_code & ec);
```

This function is used to read a certain number of bytes of data from a stream. The call will block until one of the following conditions is true:

- The supplied buffer is full (that is, it has reached maximum size).
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's read_some function.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

s The stream from which the data is to be read. The type must support the SyncReadStream concept.

b The `basic_streambuf` object into which the data will be read.

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes transferred.

Remarks

This overload is equivalent to calling:

```
boost::asio::read(
    s, b,
    boost::asio::transfer_all(), ec);
```

read (7 of 8 overloads)

Attempt to read a certain amount of data from a stream before returning.

```
template<
    typename SyncReadStream,
    typename Allocator,
    typename CompletionCondition>
std::size_t read(
    SyncReadStream & s,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition);
```

This function is used to read a certain number of bytes of data from a stream. The call will block until one of the following conditions is true:

- The supplied buffer is full (that is, it has reached maximum size).
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the stream's read_some function.

Parameters

s The stream from which the data is to be read. The type must support the SyncReadStream concept.

b The `basic_streambuf` object into which the data will be read.

completion_condition The function object to be called to determine whether the read operation is complete. The signature of the function object must be:

```
std::size_t completion_condition(
  // Result of latest read_some operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

A return value of 0 indicates that the read operation is complete. A non-zero return value indicates the maximum number of bytes to be read on the next call to the stream's read_some function.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error	Thrown on failure.

read (8 of 8 overloads)

Attempt to read a certain amount of data from a stream before returning.

```
template<
    typename SyncReadStream,
    typename Allocator,
    typename CompletionCondition>
std::size_t read(
    SyncReadStream & s,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
```

This function is used to read a certain number of bytes of data from a stream. The call will block until one of the following conditions is true:

- The supplied buffer is full (that is, it has reached maximum size).
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the stream's read_some function.

Parameters

s	The stream from which the data is to be read. The type must support the SyncReadStream concept.
b	The `basic_streambuf` object into which the data will be read.

completion_condition	The function object to be called to determine whether the read operation is complete. The signature of the function object must be:
	
	`std::size_t completion_condition(`
	`  // Result of latest read_some operation.`
	`  const boost::system::error_code& error,`
	
	`  // Number of bytes transferred so far.`
	`  std::size_t bytes_transferred`
	`);`
	
	A return value of 0 indicates that the read operation is complete. A non-zero return value indicates the maximum number of bytes to be read on the next call to the stream's read_some function.

ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes read. If an error occurs, returns the total number of bytes successfully transferred prior to the error.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

read_at

Attempt to read a certain amount of data at the specified offset before returning.

```
template<
    typename SyncRandomAccessReadDevice,
    typename MutableBufferSequence>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    const MutableBufferSequence & buffers);
  » more...

template<
    typename SyncRandomAccessReadDevice,
    typename MutableBufferSequence>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...

template<
    typename SyncRandomAccessReadDevice,
    typename MutableBufferSequence,
    typename CompletionCondition>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    CompletionCondition completion_condition);
  » more...

template<
    typename SyncRandomAccessReadDevice,
    typename MutableBufferSequence,
    typename CompletionCondition>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
  » more...

template<
    typename SyncRandomAccessReadDevice,
    typename Allocator>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b);
  » more...

template<
    typename SyncRandomAccessReadDevice,
    typename Allocator>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    boost::system::error_code & ec);
  » more...

template<
    typename SyncRandomAccessReadDevice,
    typename Allocator,
    typename CompletionCondition>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition);
  » more...

template<
    typename SyncRandomAccessReadDevice,
    typename Allocator,
    typename CompletionCondition>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
  » more...
```

Requirements

Header: `boost/asio/read_at.hpp`

Convenience header: `boost/asio.hpp`

read_at (1 of 8 overloads)

Attempt to read a certain amount of data at the specified offset before returning.

```
template<
    typename SyncRandomAccessReadDevice,
    typename MutableBufferSequence>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    const MutableBufferSequence & buffers);
```

This function is used to read a certain number of bytes of data from a random access device at the specified offset. The call will block until one of the following conditions is true:

- The supplied buffers are full. That is, the bytes transferred is equal to the sum of the buffer sizes.
- An error occurred.

This operation is implemented in terms of zero or more calls to the device's read_some_at function.

Parameters

d	The device from which the data is to be read. The type must support the SyncRandomAccessReadDevice concept.
offset	The offset at which the data will be read.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffers One or more buffers into which the data will be read. The sum of the buffer sizes indicates the maximum number of bytes to read from the device.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error Thrown on failure.

Example

To read into a single data buffer use the `buffer` function as follows:

```
boost::asio::read_at(d, 42, boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

Remarks

This overload is equivalent to calling:

```
boost::asio::read_at(
    d, 42, buffers,
    boost::asio::transfer_all());
```

read_at (2 of 8 overloads)

Attempt to read a certain amount of data at the specified offset before returning.

```
template<
    typename SyncRandomAccessReadDevice,
    typename MutableBufferSequence>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to read a certain number of bytes of data from a random access device at the specified offset. The call will block until one of the following conditions is true:

- The supplied buffers are full. That is, the bytes transferred is equal to the sum of the buffer sizes.
- An error occurred.

This operation is implemented in terms of zero or more calls to the device's read_some_at function.

Parameters

d The device from which the data is to be read. The type must support the SyncRandomAccessReadDevice concept.

offset The offset at which the data will be read.

buffers One or more buffers into which the data will be read. The sum of the buffer sizes indicates the maximum number of bytes to read from the device.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes transferred.

Example

To read into a single data buffer use the `buffer` function as follows:

```
boost::asio::read_at(d, 42,
    boost::asio::buffer(data, size), ec);
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

Remarks

This overload is equivalent to calling:

```
boost::asio::read_at(
    d, 42, buffers,
    boost::asio::transfer_all(), ec);
```

read_at (3 of 8 overloads)

Attempt to read a certain amount of data at the specified offset before returning.

```
template<
    typename SyncRandomAccessReadDevice,
    typename MutableBufferSequence,
    typename CompletionCondition>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    CompletionCondition completion_condition);
```

This function is used to read a certain number of bytes of data from a random access device at the specified offset. The call will block until one of the following conditions is true:

- The supplied buffers are full. That is, the bytes transferred is equal to the sum of the buffer sizes.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the device's read_some_at function.

Parameters

d	The device from which the data is to be read. The type must support the SyncRandomAccessReadDevice concept.
offset	The offset at which the data will be read.
buffers	One or more buffers into which the data will be read. The sum of the buffer sizes indicates the maximum number of bytes to read from the device.
completion_condition	The function object to be called to determine whether the read operation is complete. The signature of the function object must be:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t completion_condition(
  // Result of latest read_some_at operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the read operation is complete. A non-zero return value indicates the maximum number of bytes to be read on the next call to the device's read_some_at function.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error	Thrown on failure.

Example

To read into a single data buffer use the `buffer` function as follows:

```
boost::asio::read_at(d, 42, boost::asio::buffer(data, size),
    boost::asio::transfer_at_least(32));
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

read_at (4 of 8 overloads)

Attempt to read a certain amount of data at the specified offset before returning.

```
template<
    typename SyncRandomAccessReadDevice,
    typename MutableBufferSequence,
    typename CompletionCondition>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
```

This function is used to read a certain number of bytes of data from a random access device at the specified offset. The call will block until one of the following conditions is true:

- The supplied buffers are full. That is, the bytes transferred is equal to the sum of the buffer sizes.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the device's read_some_at function.

Parameters

d	The device from which the data is to be read. The type must support the SyncRandomAccessReadDevice concept.
offset	The offset at which the data will be read.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffers — One or more buffers into which the data will be read. The sum of the buffer sizes indicates the maximum number of bytes to read from the device.

completion_condition — The function object to be called to determine whether the read operation is complete. The signature of the function object must be:

```
std::size_t completion_condition(
  // Result of latest read_some_at operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the read operation is complete. A non-zero return value indicates the maximum number of bytes to be read on the next call to the device's read_some_at function.

ec — Set to indicate what error occurred, if any.

Return Value

The number of bytes read. If an error occurs, returns the total number of bytes successfully transferred prior to the error.

read_at (5 of 8 overloads)

Attempt to read a certain amount of data at the specified offset before returning.

```
template<
    typename SyncRandomAccessReadDevice,
    typename Allocator>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b);
```

This function is used to read a certain number of bytes of data from a random access device at the specified offset. The call will block until one of the following conditions is true:

- An error occurred.

This operation is implemented in terms of zero or more calls to the device's read_some_at function.

Parameters

d — The device from which the data is to be read. The type must support the SyncRandomAccessReadDevice concept.

offset — The offset at which the data will be read.

b — The `basic_streambuf` object into which the data will be read.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error — Thrown on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

This overload is equivalent to calling:

```
boost::asio::read_at(
    d, 42, b,
    boost::asio::transfer_all());
```

read_at (6 of 8 overloads)

Attempt to read a certain amount of data at the specified offset before returning.

```
template<
    typename SyncRandomAccessReadDevice,
    typename Allocator>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    boost::system::error_code & ec);
```

This function is used to read a certain number of bytes of data from a random access device at the specified offset. The call will block until one of the following conditions is true:

- An error occurred.

This operation is implemented in terms of zero or more calls to the device's read_some_at function.

Parameters

d	The device from which the data is to be read. The type must support the SyncRandomAccessReadDevice concept.
offset	The offset at which the data will be read.
b	The `basic_streambuf` object into which the data will be read.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes transferred.

Remarks

This overload is equivalent to calling:

```
boost::asio::read_at(
    d, 42, b,
    boost::asio::transfer_all(), ec);
```

read_at (7 of 8 overloads)

Attempt to read a certain amount of data at the specified offset before returning.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename SyncRandomAccessReadDevice,
    typename Allocator,
    typename CompletionCondition>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition);
```

This function is used to read a certain number of bytes of data from a random access device at the specified offset. The call will block until one of the following conditions is true:

- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the device's read_some_at function.

Parameters

d	The device from which the data is to be read. The type must support the SyncRandomAccessReadDevice concept.
offset	The offset at which the data will be read.
b	The `basic_streambuf` object into which the data will be read.
completion_condition	The function object to be called to determine whether the read operation is complete. The signature of the function object must be:

```
std::size_t completion_condition(
  // Result of latest read_some_at operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the read operation is complete. A non-zero return value indicates the maximum number of bytes to be read on the next call to the device's read_some_at function.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error	Thrown on failure.

read_at (8 of 8 overloads)

Attempt to read a certain amount of data at the specified offset before returning.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename SyncRandomAccessReadDevice,
    typename Allocator,
    typename CompletionCondition>
std::size_t read_at(
    SyncRandomAccessReadDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
```

This function is used to read a certain number of bytes of data from a random access device at the specified offset. The call will block until one of the following conditions is true:

- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the device's read_some_at function.

Parameters

d	The device from which the data is to be read. The type must support the SyncRandomAccessReadDevice concept.
offset	The offset at which the data will be read.
b	The `basic_streambuf` object into which the data will be read.

completion_condition	The function object to be called to determine whether the read operation is complete. The signature of the function object must be:
	`std::size_t completion_condition(`
	`  // Result of latest read_some_at operation.`
	`  const boost::system::error_code& error,`
	`  // Number of bytes transferred so far.`
	`  std::size_t bytes_transferred`
	`);`
	A return value of 0 indicates that the read operation is complete. A non-zero return value indicates the maximum number of bytes to be read on the next call to the device's read_some_at function.

ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes read. If an error occurs, returns the total number of bytes successfully transferred prior to the error.

read_until

Read data into a streambuf until it contains a delimiter, matches a regular expression, or a function object indicates a match.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    char delim);
  » more...

template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    char delim,
    boost::system::error_code & ec);
  » more...

template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    const std::string & delim);
  » more...

template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    const std::string & delim,
    boost::system::error_code & ec);
  » more...

template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    const boost::regex & expr);
  » more...

template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    const boost::regex & expr,
    boost::system::error_code & ec);
  » more...

template<
    typename SyncReadStream,
    typename Allocator,
    typename MatchCondition>
std::size_t read_until(
    SyncReadStream & s,
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    boost::asio::basic_streambuf< Allocator > & b,
    MatchCondition match_condition,
    typename boost::enable_if< is_match_condition< MatchCondition > >::type *  = 0);
  » more...

template<
    typename SyncReadStream,
    typename Allocator,
    typename MatchCondition>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    MatchCondition match_condition,
    boost::system::error_code & ec,
    typename boost::enable_if< is_match_condition< MatchCondition > >::type *  = 0);
  » more...
```

Requirements

Header: `boost/asio/read_until.hpp`

Convenience header: `boost/asio.hpp`

read_until (1 of 8 overloads)

Read data into a streambuf until it contains a specified delimiter.

```
template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    char delim);
```

This function is used to read data into the specified streambuf until the streambuf's get area contains the specified delimiter. The call will block until one of the following conditions is true:

- The get area of the streambuf contains the specified delimiter.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's read_some function. If the streambuf's get area already contains the delimiter, the function returns immediately.

Parameters

s	The stream from which the data is to be read. The type must support the SyncReadStream concept.
b	A streambuf object into which the data will be read.
delim	The delimiter character.

Return Value

The number of bytes in the streambuf's get area up to and including the delimiter.

Exceptions

boost::system::system_error	Thrown on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

After a successful read_until operation, the streambuf may contain additional data beyond the delimiter. An application will typically leave that data in the streambuf for a subsequent read_until operation to examine.

Example

To read data into a streambuf until a newline is encountered:

```
boost::asio::streambuf b;
boost::asio::read_until(s, b, '\n');
std::istream is(&b);
std::string line;
std::getline(is, line);
```

After the `read_until` operation completes successfully, the buffer `b` contains the delimiter:

```
{ 'a', 'b', ..., 'c', '\n', 'd', 'e', ... }
```

The call to `std::getline` then extracts the data up to and including the delimiter, so that the string `line` contains:

```
{ 'a', 'b', ..., 'c', '\n' }
```

The remaining data is left in the buffer `b` as follows:

```
{ 'd', 'e', ... }
```

This data may be the start of a new line, to be extracted by a subsequent `read_until` operation.

read_until (2 of 8 overloads)

Read data into a streambuf until it contains a specified delimiter.

```
template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    char delim,
    boost::system::error_code & ec);
```

This function is used to read data into the specified streambuf until the streambuf's get area contains the specified delimiter. The call will block until one of the following conditions is true:

- The get area of the streambuf contains the specified delimiter.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's read_some function. If the streambuf's get area already contains the delimiter, the function returns immediately.

Parameters

s The stream from which the data is to be read. The type must support the SyncReadStream concept.

b A streambuf object into which the data will be read.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

delim The delimiter character.

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes in the streambuf's get area up to and including the delimiter. Returns 0 if an error occurred.

Remarks

After a successful read_until operation, the streambuf may contain additional data beyond the delimiter. An application will typically leave that data in the streambuf for a subsequent read_until operation to examine.

read_until (3 of 8 overloads)

Read data into a streambuf until it contains a specified delimiter.

```
template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    const std::string & delim);
```

This function is used to read data into the specified streambuf until the streambuf's get area contains the specified delimiter. The call will block until one of the following conditions is true:

- The get area of the streambuf contains the specified delimiter.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's read_some function. If the streambuf's get area already contains the delimiter, the function returns immediately.

Parameters

s The stream from which the data is to be read. The type must support the SyncReadStream concept.

b A streambuf object into which the data will be read.

delim The delimiter string.

Return Value

The number of bytes in the streambuf's get area up to and including the delimiter.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

After a successful read_until operation, the streambuf may contain additional data beyond the delimiter. An application will typically leave that data in the streambuf for a subsequent read_until operation to examine.

Example

To read data into a streambuf until a newline is encountered:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::streambuf b;
boost::asio::read_until(s, b, "\r\n");
std::istream is(&b);
std::string line;
std::getline(is, line);
```

After the `read_until` operation completes successfully, the buffer `b` contains the delimiter:

```
{ 'a', 'b', ..., 'c', '\r', '\n', 'd', 'e', ... }
```

The call to `std::getline` then extracts the data up to and including the delimiter, so that the string `line` contains:

```
{ 'a', 'b', ..., 'c', '\r', '\n' }
```

The remaining data is left in the buffer `b` as follows:

```
{ 'd', 'e', ... }
```

This data may be the start of a new line, to be extracted by a subsequent `read_until` operation.

read_until (4 of 8 overloads)

Read data into a streambuf until it contains a specified delimiter.

```
template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    const std::string & delim,
    boost::system::error_code & ec);
```

This function is used to read data into the specified streambuf until the streambuf's get area contains the specified delimiter. The call will block until one of the following conditions is true:

- The get area of the streambuf contains the specified delimiter.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's read_some function. If the streambuf's get area already contains the delimiter, the function returns immediately.

Parameters

s	The stream from which the data is to be read. The type must support the SyncReadStream concept.
b	A streambuf object into which the data will be read.
delim	The delimiter string.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes in the streambuf's get area up to and including the delimiter. Returns 0 if an error occurred.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

After a successful read_until operation, the streambuf may contain additional data beyond the delimiter. An application will typically leave that data in the streambuf for a subsequent read_until operation to examine.

read_until (5 of 8 overloads)

Read data into a streambuf until some part of the data it contains matches a regular expression.

```
template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    const boost::regex & expr);
```

This function is used to read data into the specified streambuf until the streambuf's get area contains some data that matches a regular expression. The call will block until one of the following conditions is true:

- A substring of the streambuf's get area matches the regular expression.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's read_some function. If the streambuf's get area already contains data that matches the regular expression, the function returns immediately.

Parameters

s The stream from which the data is to be read. The type must support the SyncReadStream concept.

b A streambuf object into which the data will be read.

expr The regular expression.

Return Value

The number of bytes in the streambuf's get area up to and including the substring that matches the regular expression.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

After a successful read_until operation, the streambuf may contain additional data beyond that which matched the regular expression. An application will typically leave that data in the streambuf for a subsequent read_until operation to examine.

Example

To read data into a streambuf until a CR-LF sequence is encountered:

```
boost::asio::streambuf b;
boost::asio::read_until(s, b, boost::regex("\r\n"));
std::istream is(&b);
std::string line;
std::getline(is, line);
```

After the `read_until` operation completes successfully, the buffer `b` contains the data which matched the regular expression:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
{ 'a', 'b', ..., 'c', '\r', '\n', 'd', 'e', ... }
```

The call to `std::getline` then extracts the data up to and including the match, so that the string `line` contains:

```
{ 'a', 'b', ..., 'c', '\r', '\n' }
```

The remaining data is left in the buffer `b` as follows:

```
{ 'd', 'e', ... }
```

This data may be the start of a new line, to be extracted by a subsequent `read_until` operation.

read_until (6 of 8 overloads)

Read data into a streambuf until some part of the data it contains matches a regular expression.

```
template<
    typename SyncReadStream,
    typename Allocator>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    const boost::regex & expr,
    boost::system::error_code & ec);
```

This function is used to read data into the specified streambuf until the streambuf's get area contains some data that matches a regular expression. The call will block until one of the following conditions is true:

- A substring of the streambuf's get area matches the regular expression.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's read_some function. If the streambuf's get area already contains data that matches the regular expression, the function returns immediately.

Parameters

s The stream from which the data is to be read. The type must support the SyncReadStream concept.

b A streambuf object into which the data will be read.

expr The regular expression.

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes in the streambuf's get area up to and including the substring that matches the regular expression. Returns 0 if an error occurred.

Remarks

After a successful read_until operation, the streambuf may contain additional data beyond that which matched the regular expression. An application will typically leave that data in the streambuf for a subsequent read_until operation to examine.

read_until (7 of 8 overloads)

Read data into a streambuf until a function object indicates a match.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename SyncReadStream,
    typename Allocator,
    typename MatchCondition>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    MatchCondition match_condition,
    typename boost::enable_if< is_match_condition< MatchCondition > >::type *  = 0);
```

This function is used to read data into the specified streambuf until a user-defined match condition function object, when applied to the data contained in the streambuf, indicates a successful match. The call will block until one of the following conditions is true:

- The match condition function object returns a std::pair where the second element evaluates to true.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's read_some function. If the match condition function object already indicates a match, the function returns immediately.

Parameters

s	The stream from which the data is to be read. The type must support the SyncReadStream concept.
b	A streambuf object into which the data will be read.
match_condition	The function object to be called to determine whether a match exists. The signature of the function object must be:

```
pair<iterator, bool> match_condition(iterator begin, iterator end);
```

where `iterator` represents the type:

```
buffers_iterator<basic_streambuf<Allocator>::const_buffers_type>
```

The iterator parameters `begin` and `end` define the range of bytes to be scanned to determine whether there is a match. The `first` member of the return value is an iterator marking one-past-the-end of the bytes that have been consumed by the match function. This iterator is used to calculate the `begin` parameter for any subsequent invocation of the match condition. The `second` member of the return value is true if a match has been found, false otherwise.

Return Value

The number of bytes in the streambuf's get area that have been fully consumed by the match function.

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

After a successful read_until operation, the streambuf may contain additional data beyond that which matched the function object. An application will typically leave that data in the streambuf for a subsequent

The default implementation of the `is_match_condition` type trait evaluates to true for function pointers and function objects with a `result_type` typedef. It must be specialised for other user-defined function objects.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Examples

To read data into a streambuf until whitespace is encountered:

```
typedef boost::asio::buffers_iterator<
    boost::asio::streambuf::const_buffers_type> iterator;

std::pair<iterator, bool>
match_whitespace(iterator begin, iterator end)
{
  iterator i = begin;
  while (i != end)
    if (std::isspace(*i++))
      return std::make_pair(i, true);
  return std::make_pair(i, false);
}
...
boost::asio::streambuf b;
boost::asio::read_until(s, b, match_whitespace);
```

To read data into a streambuf until a matching character is found:

```
class match_char
{
public:
  explicit match_char(char c) : c_(c) {}

  template <typename Iterator>
  std::pair<Iterator, bool> operator()(
      Iterator begin, Iterator end) const
  {
    Iterator i = begin;
    while (i != end)
      if (c_ == *i++)
        return std::make_pair(i, true);
    return std::make_pair(i, false);
  }

private:
  char c_;
};

namespace asio {
  template <> struct is_match_condition<match_char>
    : public boost::true_type {};
} // namespace asio
...
boost::asio::streambuf b;
boost::asio::read_until(s, b, match_char('a'));
```

read_until (8 of 8 overloads)

Read data into a streambuf until a function object indicates a match.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename SyncReadStream,
    typename Allocator,
    typename MatchCondition>
std::size_t read_until(
    SyncReadStream & s,
    boost::asio::basic_streambuf< Allocator > & b,
    MatchCondition match_condition,
    boost::system::error_code & ec,
    typename boost::enable_if< is_match_condition< MatchCondition > >::type *  = 0);
```

This function is used to read data into the specified streambuf until a user-defined match condition function object, when applied to the data contained in the streambuf, indicates a successful match. The call will block until one of the following conditions is true:

- The match condition function object returns a std::pair where the second element evaluates to true.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's read_some function. If the match condition function object already indicates a match, the function returns immediately.

Parameters

s — The stream from which the data is to be read. The type must support the SyncReadStream concept.

b — A streambuf object into which the data will be read.

match_condition — The function object to be called to determine whether a match exists. The signature of the function object must be:

```
pair<iterator, bool> match_condition(iterator begin, iterator end);
```

where `iterator` represents the type:

```
buffers_iterator<basic_streambuf<Allocator>::const_buffers_type>
```

The iterator parameters `begin` and `end` define the range of bytes to be scanned to determine whether there is a match. The `first` member of the return value is an iterator marking one-past-the-end of the bytes that have been consumed by the match function. This iterator is used to calculate the `begin` parameter for any subsequent invocation of the match condition. The `second` member of the return value is true if a match has been found, false otherwise.

ec — Set to indicate what error occurred, if any.

Return Value

The number of bytes in the streambuf's get area that have been fully consumed by the match function. Returns 0 if an error occurred.

Remarks

After a successful read_until operation, the streambuf may contain additional data beyond that which matched the function object. An application will typically leave that data in the streambuf for a subsequent

The default implementation of the `is_match_condition` type trait evaluates to true for function pointers and function objects with a `result_type` typedef. It must be specialised for other user-defined function objects.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

seq_packet_socket_service

Default service implementation for a sequenced packet socket.

```
template<
    typename Protocol>
class seq_packet_socket_service :
  public io_service::service
```

Types

Name	Description
endpoint_type	The endpoint type.
implementation_type	The type of a sequenced packet socket implementation.
native_handle_type	The native socket type.
native_type	(Deprecated: Use native_handle_type.) The native socket type.
protocol_type	The protocol type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to a sequenced packet socket.
async_connect	Start an asynchronous connect.
async_receive	Start an asynchronous receive.
async_send	Start an asynchronous send.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
bind	Bind the sequenced packet socket to the specified local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close a sequenced packet socket implementation.
connect	Connect the sequenced packet socket to the specified endpoint.
construct	Construct a new sequenced packet socket implementation.
destroy	Destroy a sequenced packet socket implementation.
get_io_service	Get the io_service object that owns the service.
get_option	Get a socket option.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint.
move_assign	Move-assign from another sequenced packet socket implementation.
move_construct	Move-construct a new sequenced packet socket implementation.
native	(Deprecated: Use native_handle().) Get the native socket implementation.
native_handle	Get the native socket implementation.
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open a sequenced packet socket.
receive	Receive some data from the peer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
remote_endpoint	Get the remote endpoint.
send	Send the given data to the peer.
seq_packet_socket_service	Construct a new sequenced packet socket service for the specified io_service.
set_option	Set a socket option.
shutdown	Disable sends or receives on the socket.

Data Members

Name	Description
id	The unique service identifier.

Requirements

Header: `boost/asio/seq_packet_socket_service.hpp`

Convenience header: `boost/asio.hpp`

seq_packet_socket_service::assign

Assign an existing native socket to a sequenced packet socket.

```
boost::system::error_code assign(
    implementation_type & impl,
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
```

seq_packet_socket_service::async_connect

Start an asynchronous connect.

```
template<
    typename ConnectHandler>
void async_connect(
    implementation_type & impl,
    const endpoint_type & peer_endpoint,
    ConnectHandler handler);
```

seq_packet_socket_service::async_receive

Start an asynchronous receive.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    socket_base::message_flags in_flags,
    socket_base::message_flags & out_flags,
    ReadHandler handler);
```

seq_packet_socket_service::async_send

Start an asynchronous send.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    WriteHandler handler);
```

seq_packet_socket_service::at_mark

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

seq_packet_socket_service::available

Determine the number of bytes available for reading.

```
std::size_t available(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

seq_packet_socket_service::bind

Bind the sequenced packet socket to the specified local endpoint.

```
boost::system::error_code bind(
    implementation_type & impl,
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
```

seq_packet_socket_service::cancel

Cancel all asynchronous operations associated with the socket.

```
boost::system::error_code cancel(
    implementation_type & impl,
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

seq_packet_socket_service::close

Close a sequenced packet socket implementation.

```
boost::system::error_code close(
    implementation_type & impl,
    boost::system::error_code & ec);
```

seq_packet_socket_service::connect

Connect the sequenced packet socket to the specified endpoint.

```
boost::system::error_code connect(
    implementation_type & impl,
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
```

seq_packet_socket_service::construct

Construct a new sequenced packet socket implementation.

```
void construct(
    implementation_type & impl);
```

seq_packet_socket_service::destroy

Destroy a sequenced packet socket implementation.

```
void destroy(
    implementation_type & impl);
```

seq_packet_socket_service::endpoint_type

The endpoint type.

```
typedef Protocol::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/seq_packet_socket_service.hpp`

Convenience header: `boost/asio.hpp`

seq_packet_socket_service::get_io_service

Inherited from io_service.

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

seq_packet_socket_service::get_option

Get a socket option.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename GettableSocketOption>
boost::system::error_code get_option(
    const implementation_type & impl,
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
```

seq_packet_socket_service::id

The unique service identifier.

```
static boost::asio::io_service::id id;
```

seq_packet_socket_service::implementation_type

The type of a sequenced packet socket implementation.

```
typedef implementation_defined implementation_type;
```

Requirements

Header: `boost/asio/seq_packet_socket_service.hpp`

Convenience header: `boost/asio.hpp`

seq_packet_socket_service::io_control

Perform an IO control command on the socket.

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    implementation_type & impl,
    IoControlCommand & command,
    boost::system::error_code & ec);
```

seq_packet_socket_service::is_open

Determine whether the socket is open.

```
bool is_open(
    const implementation_type & impl) const;
```

seq_packet_socket_service::local_endpoint

Get the local endpoint.

```
endpoint_type local_endpoint(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

seq_packet_socket_service::move_assign

Move-assign from another sequenced packet socket implementation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void move_assign(
    implementation_type & impl,
    seq_packet_socket_service & other_service,
    implementation_type & other_impl);
```

seq_packet_socket_service::move_construct

Move-construct a new sequenced packet socket implementation.

```
void move_construct(
    implementation_type & impl,
    implementation_type & other_impl);
```

seq_packet_socket_service::native

(Deprecated: Use `native_handle()`.) Get the native socket implementation.

```
native_type native(
    implementation_type & impl);
```

seq_packet_socket_service::native_handle

Get the native socket implementation.

```
native_handle_type native_handle(
    implementation_type & impl);
```

seq_packet_socket_service::native_handle_type

The native socket type.

```
typedef implementation_defined native_handle_type;
```

Requirements

Header: `boost/asio/seq_packet_socket_service.hpp`

Convenience header: `boost/asio.hpp`

seq_packet_socket_service::native_non_blocking

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking(
    const implementation_type & impl) const;
  » more...
```

Sets the non-blocking mode of the native socket implementation.

```
boost::system::error_code native_non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

seq_packet_socket_service::native_non_blocking (1 of 2 overloads)

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking(
    const implementation_type & impl) const;
```

seq_packet_socket_service::native_non_blocking (2 of 2 overloads)

Sets the non-blocking mode of the native socket implementation.

```
boost::system::error_code native_non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
```

seq_packet_socket_service::native_type

(Deprecated: Use native_handle_type.) The native socket type.

```
typedef implementation_defined native_type;
```

Requirements

Header: `boost/asio/seq_packet_socket_service.hpp`

Convenience header: `boost/asio.hpp`

seq_packet_socket_service::non_blocking

Gets the non-blocking mode of the socket.

```
bool non_blocking(
    const implementation_type & impl) const;
  » more...
```

Sets the non-blocking mode of the socket.

```
boost::system::error_code non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
  » more...
```

seq_packet_socket_service::non_blocking (1 of 2 overloads)

Gets the non-blocking mode of the socket.

```
bool non_blocking(
    const implementation_type & impl) const;
```

seq_packet_socket_service::non_blocking (2 of 2 overloads)

Sets the non-blocking mode of the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
```

seq_packet_socket_service::open

Open a sequenced packet socket.

```
boost::system::error_code open(
    implementation_type & impl,
    const protocol_type & protocol,
    boost::system::error_code & ec);
```

seq_packet_socket_service::protocol_type

The protocol type.

```
typedef Protocol protocol_type;
```

Requirements

Header: `boost/asio/seq_packet_socket_service.hpp`

Convenience header: `boost/asio.hpp`

seq_packet_socket_service::receive

Receive some data from the peer.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    socket_base::message_flags in_flags,
    socket_base::message_flags & out_flags,
    boost::system::error_code & ec);
```

seq_packet_socket_service::remote_endpoint

Get the remote endpoint.

```
endpoint_type remote_endpoint(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

seq_packet_socket_service::send

Send the given data to the peer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t send(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

seq_packet_socket_service::seq_packet_socket_service

Construct a new sequenced packet socket service for the specified `io_service`.

```
seq_packet_socket_service(
    boost::asio::io_service & io_service);
```

seq_packet_socket_service::set_option

Set a socket option.

```
template<
    typename SettableSocketOption>
boost::system::error_code set_option(
    implementation_type & impl,
    const SettableSocketOption & option,
    boost::system::error_code & ec);
```

seq_packet_socket_service::shutdown

Disable sends or receives on the socket.

```
boost::system::error_code shutdown(
    implementation_type & impl,
    socket_base::shutdown_type what,
    boost::system::error_code & ec);
```

serial_port

Typedef for the typical usage of a serial port.

```
typedef basic_serial_port serial_port;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_serial_port is always the lowest layer.
native_handle_type	The native representation of a serial port.
native_type	(Deprecated: Use native_handle_type.) The native representation of a serial port.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native serial port to the serial port.
async_read_some	Start an asynchronous read.
async_write_some	Start an asynchronous write.
basic_serial_port	Construct a basic_serial_port without opening it. Construct and open a basic_serial_port. Construct a basic_serial_port on an existing native serial port. Move-construct a basic_serial_port from another.
cancel	Cancel all asynchronous operations associated with the serial port.
close	Close the serial port.
get_io_service	Get the io_service associated with the object.
get_option	Get an option from the serial port.
is_open	Determine whether the serial port is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native serial port representation.
native_handle	Get the native serial port representation.
open	Open the serial port using the specified device name.
operator=	Move-assign a basic_serial_port from another.
read_some	Read some data from the serial port.
send_break	Send a break sequence to the serial port.
set_option	Set an option on the serial port.
write_some	Write some data to the serial port.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_serial_port` class template provides functionality that is common to all serial ports.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/serial_port.hpp`

Convenience header: `boost/asio.hpp`

serial_port_base

The `serial_port_base` class is used as a base for the `basic_serial_port` class template so that we have a common place to define the serial port options.

```
class serial_port_base
```

Types

Name	Description
baud_rate	Serial port option to permit changing the baud rate.
character_size	Serial port option to permit changing the character size.
flow_control	Serial port option to permit changing the flow control.
parity	Serial port option to permit changing the parity.
stop_bits	Serial port option to permit changing the number of stop bits.

Protected Member Functions

Name	Description
~serial_port_base	Protected destructor to prevent deletion through this type.

Requirements

Header: `boost/asio/serial_port_base.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio.hpp`

serial_port_base::~serial_port_base

Protected destructor to prevent deletion through this type.

```
~serial_port_base();
```

serial_port_base::baud_rate

Serial port option to permit changing the baud rate.

```
class baud_rate
```

Member Functions

Name	Description
baud_rate	
load	
store	
value	

Implements changing the baud rate for a given serial port.

Requirements

Header: `boost/asio/serial_port_base.hpp`

Convenience header: `boost/asio.hpp`

serial_port_base::baud_rate::baud_rate

```
baud_rate(
    unsigned int rate = 0);
```

serial_port_base::baud_rate::load

```
boost::system::error_code load(
    const BOOST_ASIO_OPTION_STORAGE & storage,
    boost::system::error_code & ec);
```

serial_port_base::baud_rate::store

```
boost::system::error_code store(
    BOOST_ASIO_OPTION_STORAGE & storage,
    boost::system::error_code & ec) const;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

serial_port_base::baud_rate::value

```
unsigned int value() const;
```

serial_port_base::character_size

Serial port option to permit changing the character size.

```
class character_size
```

Member Functions

Name	Description
character_size	
load	
store	
value	

Implements changing the character size for a given serial port.

Requirements

Header: `boost/asio/serial_port_base.hpp`

Convenience header: `boost/asio.hpp`

serial_port_base::character_size::character_size

```
character_size(
    unsigned int t = 8);
```

serial_port_base::character_size::load

```
boost::system::error_code load(
    const BOOST_ASIO_OPTION_STORAGE & storage,
    boost::system::error_code & ec);
```

serial_port_base::character_size::store

```
boost::system::error_code store(
    BOOST_ASIO_OPTION_STORAGE & storage,
    boost::system::error_code & ec) const;
```

serial_port_base::character_size::value

```
unsigned int value() const;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

serial_port_base::flow_control

Serial port option to permit changing the flow control.

```
class flow_control
```

Types

Name	Description
type	

Member Functions

Name	Description
flow_control	
load	
store	
value	

Implements changing the flow control for a given serial port.

Requirements

Header: `boost/asio/serial_port_base.hpp`

Convenience header: `boost/asio.hpp`

serial_port_base::flow_control::flow_control

```
flow_control(
    type t = none);
```

serial_port_base::flow_control::load

```
boost::system::error_code load(
    const BOOST_ASIO_OPTION_STORAGE & storage,
    boost::system::error_code & ec);
```

serial_port_base::flow_control::store

```
boost::system::error_code store(
    BOOST_ASIO_OPTION_STORAGE & storage,
    boost::system::error_code & ec) const;
```

serial_port_base::flow_control::type

```
enum type
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Values

none

software

hardware

serial_port_base::flow_control::value

```
type value() const;
```

serial_port_base::parity

Serial port option to permit changing the parity.

```
class parity
```

Types

Name	Description
type	

Member Functions

Name	Description
load	
parity	
store	
value	

Implements changing the parity for a given serial port.

Requirements

Header: `boost/asio/serial_port_base.hpp`

Convenience header: `boost/asio.hpp`

serial_port_base::parity::load

```
boost::system::error_code load(
    const BOOST_ASIO_OPTION_STORAGE & storage,
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

serial_port_base::parity::parity

```
parity(
    type t = none);
```

serial_port_base::parity::store

```
boost::system::error_code store(
    BOOST_ASIO_OPTION_STORAGE & storage,
    boost::system::error_code & ec) const;
```

serial_port_base::parity::type

```
enum type
```

Values

none

odd

even

serial_port_base::parity::value

```
type value() const;
```

serial_port_base::stop_bits

Serial port option to permit changing the number of stop bits.

```
class stop_bits
```

Types

Name	Description
type	

Member Functions

Name	Description
load	
stop_bits	
store	
value	

Implements changing the number of stop bits for a given serial port.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/serial_port_base.hpp`

Convenience header: `boost/asio.hpp`

serial_port_base::stop_bits::load

```
boost::system::error_code load(
    const BOOST_ASIO_OPTION_STORAGE & storage,
    boost::system::error_code & ec);
```

serial_port_base::stop_bits::stop_bits

```
stop_bits(
    type t = one);
```

serial_port_base::stop_bits::store

```
boost::system::error_code store(
    BOOST_ASIO_OPTION_STORAGE & storage,
    boost::system::error_code & ec) const;
```

serial_port_base::stop_bits::type

```
enum type
```

Values

one

onepointfive

two

serial_port_base::stop_bits::value

```
type value() const;
```

serial_port_service

Default service implementation for a serial port.

```
class serial_port_service :
  public io_service::service
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
implementation_type	The type of a serial port implementation.
native_handle_type	The native handle type.
native_type	(Deprecated: Use native_handle_type.) The native handle type.

Member Functions

Name	Description
assign	Assign an existing native handle to a serial port.
async_read_some	Start an asynchronous read.
async_write_some	Start an asynchronous write.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close a serial port implementation.
construct	Construct a new serial port implementation.
destroy	Destroy a serial port implementation.
get_io_service	Get the io_service object that owns the service.
get_option	Get a serial port option.
is_open	Determine whether the handle is open.
move_assign	Move-assign from another serial port implementation.
move_construct	Move-construct a new serial port implementation.
native	(Deprecated: Use native_handle().) Get the native handle implementation.
native_handle	Get the native handle implementation.
open	Open a serial port.
read_some	Read some data from the stream.
send_break	Send a break sequence to the serial port.
serial_port_service	Construct a new serial port service for the specified io_service.
set_option	Set a serial port option.
write_some	Write the given data to the stream.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Data Members

Name	Description
id	The unique service identifier.

Requirements

Header: `boost/asio/serial_port_service.hpp`

Convenience header: `boost/asio.hpp`

serial_port_service::assign

Assign an existing native handle to a serial port.

```
boost::system::error_code assign(
    implementation_type & impl,
    const native_handle_type & handle,
    boost::system::error_code & ec);
```

serial_port_service::async_read_some

Start an asynchronous read.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_some(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

serial_port_service::async_write_some

Start an asynchronous write.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_some(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

serial_port_service::cancel

Cancel all asynchronous operations associated with the handle.

```
boost::system::error_code cancel(
    implementation_type & impl,
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

serial_port_service::close

Close a serial port implementation.

```
boost::system::error_code close(
    implementation_type & impl,
    boost::system::error_code & ec);
```

serial_port_service::construct

Construct a new serial port implementation.

```
void construct(
    implementation_type & impl);
```

serial_port_service::destroy

Destroy a serial port implementation.

```
void destroy(
    implementation_type & impl);
```

serial_port_service::get_io_service

Inherited from io_service.

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

serial_port_service::get_option

Get a serial port option.

```
template<
    typename GettableSerialPortOption>
boost::system::error_code get_option(
    const implementation_type & impl,
    GettableSerialPortOption & option,
    boost::system::error_code & ec) const;
```

serial_port_service::id

The unique service identifier.

```
static boost::asio::io_service::id id;
```

serial_port_service::implementation_type

The type of a serial port implementation.

```
typedef implementation_defined implementation_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/serial_port_service.hpp`

Convenience header: `boost/asio.hpp`

serial_port_service::is_open

Determine whether the handle is open.

```
bool is_open(
    const implementation_type & impl) const;
```

serial_port_service::move_assign

Move-assign from another serial port implementation.

```
void move_assign(
    implementation_type & impl,
    serial_port_service & other_service,
    implementation_type & other_impl);
```

serial_port_service::move_construct

Move-construct a new serial port implementation.

```
void move_construct(
    implementation_type & impl,
    implementation_type & other_impl);
```

serial_port_service::native

(Deprecated: Use `native_handle()`.) Get the native handle implementation.

```
native_type native(
    implementation_type & impl);
```

serial_port_service::native_handle

Get the native handle implementation.

```
native_handle_type native_handle(
    implementation_type & impl);
```

serial_port_service::native_handle_type

The native handle type.

```
typedef implementation_defined native_handle_type;
```

Requirements

Header: `boost/asio/serial_port_service.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio.hpp`

serial_port_service::native_type

(Deprecated: Use native_handle_type.) The native handle type.

```
typedef implementation_defined native_type;
```

Requirements

Header: `boost/asio/serial_port_service.hpp`

Convenience header: `boost/asio.hpp`

serial_port_service::open

Open a serial port.

```
boost::system::error_code open(
    implementation_type & impl,
    const std::string & device,
    boost::system::error_code & ec);
```

serial_port_service::read_some

Read some data from the stream.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

serial_port_service::send_break

Send a break sequence to the serial port.

```
boost::system::error_code send_break(
    implementation_type & impl,
    boost::system::error_code & ec);
```

serial_port_service::serial_port_service

Construct a new serial port service for the specified `io_service`.

```
serial_port_service(
    boost::asio::io_service & io_service);
```

serial_port_service::set_option

Set a serial port option.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename SettableSerialPortOption>
boost::system::error_code set_option(
    implementation_type & impl,
    const SettableSerialPortOption & option,
    boost::system::error_code & ec);
```

serial_port_service::write_some

Write the given data to the stream.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

service_already_exists

Exception thrown when trying to add a duplicate service to an `io_service`.

```
class service_already_exists
```

Member Functions

Name	Description
service_already_exists	

Requirements

Header: `boost/asio/io_service.hpp`

Convenience header: `boost/asio.hpp`

service_already_exists::service_already_exists

```
service_already_exists();
```

signal_set

Typedef for the typical usage of a signal set.

```
typedef basic_signal_set signal_set;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
service_type	The type of the service that will be used to provide I/O operations.

Member Functions

Name	Description
add	Add a signal to a signal_set.
async_wait	Start an asynchronous operation to wait for a signal to be delivered.
basic_signal_set	Construct a signal set without adding any signals. Construct a signal set and add one signal. Construct a signal set and add two signals. Construct a signal set and add three signals.
cancel	Cancel all operations associated with the signal set.
clear	Remove all signals from a signal_set.
get_io_service	Get the io_service associated with the object.
remove	Remove a signal from a signal_set.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_signal_set` class template provides the ability to perform an asynchronous wait for one or more signals to occur.

Most applications will use the `signal_set` typedef.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Example

Performing an asynchronous wait:

```
void handler(
    const boost::system::error_code& error,
    int signal_number)
{
  if (!error)
  {
    // A signal occurred.
  }
}

...

// Construct a signal set registered for process termination.
boost::asio::signal_set signals(io_service, SIGINT, SIGTERM);

// Start an asynchronous wait for one of the signals to occur.
signals.async_wait(handler);
```

Queueing of signal notifications

If a signal is registered with a signal_set, and the signal occurs when there are no waiting handlers, then the signal notification is queued. The next async_wait operation on that signal_set will dequeue the notification. If multiple notifications are queued, subsequent async_wait operations dequeue them one at a time. Signal notifications are dequeued in order of ascending signal number.

If a signal number is removed from a signal_set (using the `remove` or `erase` member functions) then any queued notifications for that signal are discarded.

Multiple registration of signals

The same signal number may be registered with different signal_set objects. When the signal occurs, one handler is called for each signal_set object.

Note that multiple registration only works for signals that are registered using Asio. The application must not also register a signal handler using functions such as `signal()` or `sigaction()`.

Signal masking on POSIX platforms

POSIX allows signals to be blocked using functions such as `sigprocmask()` and `pthread_sigmask()`. For signals to be delivered, programs must ensure that any signals registered using signal_set objects are unblocked in at least one thread.

Requirements

Header: `boost/asio/signal_set.hpp`

Convenience header: `boost/asio.hpp`

signal_set_service

Default service implementation for a signal set.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
class signal_set_service :
  public io_service::service
```

Types

Name	Description
implementation_type	The type of a signal set implementation.

Member Functions

Name	Description
add	Add a signal to a signal_set.
async_wait	
cancel	Cancel all operations associated with the signal set.
clear	Remove all signals from a signal_set.
construct	Construct a new signal set implementation.
destroy	Destroy a signal set implementation.
get_io_service	Get the io_service object that owns the service.
remove	Remove a signal to a signal_set.
signal_set_service	Construct a new signal set service for the specified io_service.

Data Members

Name	Description
id	The unique service identifier.

Requirements

Header: `boost/asio/signal_set_service.hpp`

Convenience header: `boost/asio.hpp`

signal_set_service::add

Add a signal to a signal_set.

```
boost::system::error_code add(
    implementation_type & impl,
    int signal_number,
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

signal_set_service::async_wait

```
template<
    typename SignalHandler>
void async_wait(
    implementation_type & impl,
    SignalHandler handler);
```

signal_set_service::cancel

Cancel all operations associated with the signal set.

```
boost::system::error_code cancel(
    implementation_type & impl,
    boost::system::error_code & ec);
```

signal_set_service::clear

Remove all signals from a signal_set.

```
boost::system::error_code clear(
    implementation_type & impl,
    boost::system::error_code & ec);
```

signal_set_service::construct

Construct a new signal set implementation.

```
void construct(
    implementation_type & impl);
```

signal_set_service::destroy

Destroy a signal set implementation.

```
void destroy(
    implementation_type & impl);
```

signal_set_service::get_io_service

Inherited from io_service.

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

signal_set_service::id

The unique service identifier.

```
static boost::asio::io_service::id id;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

signal_set_service::implementation_type

The type of a signal set implementation.

```
typedef implementation_defined implementation_type;
```

Requirements

Header: `boost/asio/signal_set_service.hpp`

Convenience header: `boost/asio.hpp`

signal_set_service::remove

Remove a signal to a signal_set.

```
boost::system::error_code remove(
    implementation_type & impl,
    int signal_number,
    boost::system::error_code & ec);
```

signal_set_service::signal_set_service

Construct a new signal set service for the specified `io_service`.

```
signal_set_service(
    boost::asio::io_service & io_service);
```

socket_acceptor_service

Default service implementation for a socket acceptor.

```
template<
    typename Protocol>
class socket_acceptor_service :
  public io_service::service
```

Types

Name	Description
endpoint_type	The endpoint type.
implementation_type	The native type of the socket acceptor.
native_handle_type	The native acceptor type.
native_type	(Deprecated: Use native_handle_type.) The native acceptor type.
protocol_type	The protocol type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
accept	Accept a new connection.
assign	Assign an existing native acceptor to a socket acceptor.
async_accept	Start an asynchronous accept.
bind	Bind the socket acceptor to the specified local endpoint.
cancel	Cancel all asynchronous operations associated with the acceptor.
close	Close a socket acceptor implementation.
construct	Construct a new socket acceptor implementation.
destroy	Destroy a socket acceptor implementation.
get_io_service	Get the io_service object that owns the service.
get_option	Get a socket option.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the acceptor is open.
listen	Place the socket acceptor into the state where it will listen for new connections.
local_endpoint	Get the local endpoint.
move_assign	Move-assign from another socket acceptor implementation.
move_construct	Move-construct a new socket acceptor implementation.
native	(Deprecated: Use native_handle().) Get the native acceptor implementation.
native_handle	Get the native acceptor implementation.
native_non_blocking	Gets the non-blocking mode of the native acceptor implementation. Sets the non-blocking mode of the native acceptor implementation.
non_blocking	Gets the non-blocking mode of the acceptor. Sets the non-blocking mode of the acceptor.
open	Open a new socket acceptor implementation.
set_option	Set a socket option.
socket_acceptor_service	Construct a new socket acceptor service for the specified io_service.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Data Members

Name	Description
id	The unique service identifier.

Requirements

Header: `boost/asio/socket_acceptor_service.hpp`

Convenience header: `boost/asio.hpp`

socket_acceptor_service::accept

Accept a new connection.

```
template<
    typename SocketService>
boost::system::error_code accept(
    implementation_type & impl,
    basic_socket< protocol_type, SocketService > & peer,
    endpoint_type * peer_endpoint,
    boost::system::error_code & ec);
```

socket_acceptor_service::assign

Assign an existing native acceptor to a socket acceptor.

```
boost::system::error_code assign(
    implementation_type & impl,
    const protocol_type & protocol,
    const native_handle_type & native_acceptor,
    boost::system::error_code & ec);
```

socket_acceptor_service::async_accept

Start an asynchronous accept.

```
template<
    typename SocketService,
    typename AcceptHandler>
void async_accept(
    implementation_type & impl,
    basic_socket< protocol_type, SocketService > & peer,
    endpoint_type * peer_endpoint,
    AcceptHandler handler);
```

socket_acceptor_service::bind

Bind the socket acceptor to the specified local endpoint.

```
boost::system::error_code bind(
    implementation_type & impl,
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

socket_acceptor_service::cancel

Cancel all asynchronous operations associated with the acceptor.

```
boost::system::error_code cancel(
    implementation_type & impl,
    boost::system::error_code & ec);
```

socket_acceptor_service::close

Close a socket acceptor implementation.

```
boost::system::error_code close(
    implementation_type & impl,
    boost::system::error_code & ec);
```

socket_acceptor_service::construct

Construct a new socket acceptor implementation.

```
void construct(
    implementation_type & impl);
```

socket_acceptor_service::destroy

Destroy a socket acceptor implementation.

```
void destroy(
    implementation_type & impl);
```

socket_acceptor_service::endpoint_type

The endpoint type.

```
typedef protocol_type::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/socket_acceptor_service.hpp`

Convenience header: `boost/asio.hpp`

socket_acceptor_service::get_io_service

Inherited from io_service.

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

socket_acceptor_service::get_option

Get a socket option.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename GettableSocketOption>
boost::system::error_code get_option(
    const implementation_type & impl,
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
```

socket_acceptor_service::id

The unique service identifier.

```
static boost::asio::io_service::id id;
```

socket_acceptor_service::implementation_type

The native type of the socket acceptor.

```
typedef implementation_defined implementation_type;
```

Requirements

Header: `boost/asio/socket_acceptor_service.hpp`

Convenience header: `boost/asio.hpp`

socket_acceptor_service::io_control

Perform an IO control command on the socket.

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    implementation_type & impl,
    IoControlCommand & command,
    boost::system::error_code & ec);
```

socket_acceptor_service::is_open

Determine whether the acceptor is open.

```
bool is_open(
    const implementation_type & impl) const;
```

socket_acceptor_service::listen

Place the socket acceptor into the state where it will listen for new connections.

```
boost::system::error_code listen(
    implementation_type & impl,
    int backlog,
    boost::system::error_code & ec);
```

socket_acceptor_service::local_endpoint

Get the local endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
endpoint_type local_endpoint(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

socket_acceptor_service::move_assign

Move-assign from another socket acceptor implementation.

```
void move_assign(
    implementation_type & impl,
    socket_acceptor_service & other_service,
    implementation_type & other_impl);
```

socket_acceptor_service::move_construct

Move-construct a new socket acceptor implementation.

```
void move_construct(
    implementation_type & impl,
    implementation_type & other_impl);
```

socket_acceptor_service::native

(Deprecated: Use `native_handle()`.) Get the native acceptor implementation.

```
native_type native(
    implementation_type & impl);
```

socket_acceptor_service::native_handle

Get the native acceptor implementation.

```
native_handle_type native_handle(
    implementation_type & impl);
```

socket_acceptor_service::native_handle_type

The native acceptor type.

```
typedef implementation_defined native_handle_type;
```

Requirements

Header: `boost/asio/socket_acceptor_service.hpp`

Convenience header: `boost/asio.hpp`

socket_acceptor_service::native_non_blocking

Gets the non-blocking mode of the native acceptor implementation.

```
bool native_non_blocking(
    const implementation_type & impl) const;
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Sets the non-blocking mode of the native acceptor implementation.

```
boost::system::error_code native_non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
  » more...
```

socket_acceptor_service::native_non_blocking (1 of 2 overloads)

Gets the non-blocking mode of the native acceptor implementation.

```
bool native_non_blocking(
    const implementation_type & impl) const;
```

socket_acceptor_service::native_non_blocking (2 of 2 overloads)

Sets the non-blocking mode of the native acceptor implementation.

```
boost::system::error_code native_non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
```

socket_acceptor_service::native_type

(Deprecated: Use native_handle_type.) The native acceptor type.

```
typedef implementation_defined native_type;
```

Requirements

Header: `boost/asio/socket_acceptor_service.hpp`

Convenience header: `boost/asio.hpp`

socket_acceptor_service::non_blocking

Gets the non-blocking mode of the acceptor.

```
bool non_blocking(
    const implementation_type & impl) const;
  » more...
```

Sets the non-blocking mode of the acceptor.

```
boost::system::error_code non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
  » more...
```

socket_acceptor_service::non_blocking (1 of 2 overloads)

Gets the non-blocking mode of the acceptor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
bool non_blocking(
    const implementation_type & impl) const;
```

socket_acceptor_service::non_blocking (2 of 2 overloads)

Sets the non-blocking mode of the acceptor.

```
boost::system::error_code non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
```

socket_acceptor_service::open

Open a new socket acceptor implementation.

```
boost::system::error_code open(
    implementation_type & impl,
    const protocol_type & protocol,
    boost::system::error_code & ec);
```

socket_acceptor_service::protocol_type

The protocol type.

```
typedef Protocol protocol_type;
```

Requirements

Header: `boost/asio/socket_acceptor_service.hpp`

Convenience header: `boost/asio.hpp`

socket_acceptor_service::set_option

Set a socket option.

```
template<
    typename SettableSocketOption>
boost::system::error_code set_option(
    implementation_type & impl,
    const SettableSocketOption & option,
    boost::system::error_code & ec);
```

socket_acceptor_service::socket_acceptor_service

Construct a new socket acceptor service for the specified `io_service`.

```
socket_acceptor_service(
    boost::asio::io_service & io_service);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

socket_base

The `socket_base` class is used as a base for the `basic_stream_socket` and `basic_datagram_socket` class templates so that we have a common place to define the shutdown_type and enum.

```
class socket_base
```

Types

Name	Description
broadcast	Socket option to permit sending of broadcast messages.
bytes_readable	IO control command to get the amount of data that can be read without blocking.
debug	Socket option to enable socket-level debugging.
do_not_route	Socket option to prevent routing, use local interfaces only.
enable_connection_aborted	Socket option to report aborted connections on accept.
keep_alive	Socket option to send keep-alives.
linger	Socket option to specify whether the socket lingers on close if unsent data is present.
message_flags	Bitmask type for flags that can be passed to send and receive operations.
non_blocking_io	(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.
receive_buffer_size	Socket option for the receive buffer size of a socket.
receive_low_watermark	Socket option for the receive low watermark.
reuse_address	Socket option to allow the socket to be bound to an address that is already in use.
send_buffer_size	Socket option for the send buffer size of a socket.
send_low_watermark	Socket option for the send low watermark.
shutdown_type	Different ways a socket may be shutdown.

Protected Member Functions

Name	Description
~socket_base	Protected destructor to prevent deletion through this type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Data Members

Name	Description
max_connections	The maximum length of the queue of pending incoming connections.
message_do_not_route	Specify that the data should not be subject to routing.
message_end_of_record	Specifies that the data marks the end of a record.
message_out_of_band	Process out-of-band data.
message_peek	Peek at incoming data without removing it from the input queue.

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

socket_base::broadcast

Socket option to permit sending of broadcast messages.

```
typedef implementation_defined broadcast;
```

Implements the SOL_SOCKET/SO_BROADCAST socket option.

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::broadcast option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

socket_base::bytes_readable

IO control command to get the amount of data that can be read without blocking.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef implementation_defined bytes_readable;
```

Implements the FIONREAD IO control command.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::bytes_readable command(true);
socket.io_control(command);
std::size_t bytes_readable = command.get();
```

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

socket_base::debug

Socket option to enable socket-level debugging.

```
typedef implementation_defined debug;
```

Implements the SOL_SOCKET/SO_DEBUG socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::debug option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

socket_base::do_not_route

Socket option to prevent routing, use local interfaces only.

```
typedef implementation_defined do_not_route;
```

Implements the SOL_SOCKET/SO_DONTROUTE socket option.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Examples

Setting the option:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::udp::socket socket(io_service);
...
boost::asio::socket_base::do_not_route option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

socket_base::enable_connection_aborted

Socket option to report aborted connections on accept.

```
typedef implementation_defined enable_connection_aborted;
```

Implements a custom socket option that determines whether or not an accept operation is permitted to fail with `boost::asio::error::connection_aborted`. By default the option is false.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option(true);
acceptor.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::enable_connection_aborted option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

socket_base::keep_alive

Socket option to send keep-alives.

```
typedef implementation_defined keep_alive;
```

Implements the SOL_SOCKET/SO_KEEPALIVE socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option(true);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::keep_alive option;
socket.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

socket_base::linger

Socket option to specify whether the socket lingers on close if unsent data is present.

```
typedef implementation_defined linger;
```

Implements the SOL_SOCKET/SO_LINGER socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option(true, 30);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::linger option;
socket.get_option(option);
bool is_set = option.enabled();
unsigned short timeout = option.timeout();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

socket_base::max_connections

The maximum length of the queue of pending incoming connections.

```
static const int max_connections = implementation_defined;
```

socket_base::message_do_not_route

Specify that the data should not be subject to routing.

```
static const int message_do_not_route = implementation_defined;
```

socket_base::message_end_of_record

Specifies that the data marks the end of a record.

```
static const int message_end_of_record = implementation_defined;
```

socket_base::message_flags

Bitmask type for flags that can be passed to send and receive operations.

```
typedef int message_flags;
```

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

socket_base::message_out_of_band

Process out-of-band data.

```
static const int message_out_of_band = implementation_defined;
```

socket_base::message_peek

Peek at incoming data without removing it from the input queue.

```
static const int message_peek = implementation_defined;
```

socket_base::non_blocking_io

(Deprecated: Use non_blocking().) IO control command to set the blocking mode of the socket.

```
typedef implementation_defined non_blocking_io;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Implements the FIONBIO IO control command.

Example

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::non_blocking_io command(true);
socket.io_control(command);
```

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

socket_base::receive_buffer_size

Socket option for the receive buffer size of a socket.

```
typedef implementation_defined receive_buffer_size;
```

Implements the SOL_SOCKET/SO_RCVBUF socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

socket_base::receive_low_watermark

Socket option for the receive low watermark.

```
typedef implementation_defined receive_low_watermark;
```

Implements the SOL_SOCKET/SO_RCVLOWAT socket option.

Examples

Setting the option:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::receive_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

socket_base::reuse_address

Socket option to allow the socket to be bound to an address that is already in use.

```
typedef implementation_defined reuse_address;
```

Implements the SOL_SOCKET/SO_REUSEADDR socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option(true);
acceptor.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::acceptor acceptor(io_service);
...
boost::asio::socket_base::reuse_address option;
acceptor.get_option(option);
bool is_set = option.value();
```

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

socket_base::send_buffer_size

Socket option for the send buffer size of a socket.

```
typedef implementation_defined send_buffer_size;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Implements the SOL_SOCKET/SO_SNDBUF socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option(8192);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_buffer_size option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

socket_base::send_low_watermark

Socket option for the send low watermark.

```
typedef implementation_defined send_low_watermark;
```

Implements the SOL_SOCKET/SO_SNDLOWAT socket option.

Examples

Setting the option:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option(1024);
socket.set_option(option);
```

Getting the current option value:

```
boost::asio::ip::tcp::socket socket(io_service);
...
boost::asio::socket_base::send_low_watermark option;
socket.get_option(option);
int size = option.value();
```

Requirements

Header: `boost/asio/socket_base.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

socket_base::shutdown_type

Different ways a socket may be shutdown.

```
enum shutdown_type
```

Values

shutdown_receive	Shutdown the receive side of the socket.
shutdown_send	Shutdown the send side of the socket.
shutdown_both	Shutdown both send and receive on the socket.

socket_base::~socket_base

Protected destructor to prevent deletion through this type.

```
~socket_base();
```

ssl::context

```
class context :
  public ssl::context_base,
  noncopyable
```

Types

Name	Description
file_format	File format types.
impl_type	(Deprecated: Use native_handle_type.) The native type of the SSL context.
method	Different methods supported by a context.
native_handle_type	The native handle type of the SSL context.
options	Bitmask type for SSL options.
password_purpose	Purpose of PEM password.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
add_verify_path	Add a directory containing certificate authority files to be used for performing verification.
context	Constructor. Deprecated constructor taking a reference to an io_service object. Move-construct a context from another.
impl	(Deprecated: Use native_handle().) Get the underlying implementation in the native type.
load_verify_file	Load a certification authority file for performing verification.
native_handle	Get the underlying implementation in the native type.
operator=	Move-assign a context from another.
set_default_verify_paths	Configures the context to use the default directories for finding certification authority certificates.
set_options	Set options on the context.
set_password_callback	Set the password callback.
set_verify_callback	Set the callback used to verify peer certificates.
set_verify_mode	Set the peer verification mode.
use_certificate_chain_file	Use a certificate chain from a file.
use_certificate_file	Use a certificate from a file.
use_private_key_file	Use a private key from a file.
use_rsa_private_key_file	Use an RSA private key from a file.
use_tmp_dh_file	Use the specified file to obtain the temporary Diffie-Hellman parameters.
~context	Destructor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Data Members

Name	Description
default_workarounds	Implement various bug workarounds.
no_sslv2	Disable SSL v2.
no_sslv3	Disable SSL v3.
no_tlsv1	Disable TLS v1.
single_dh_use	Always create a new key when using tmp_dh parameters.

Requirements

Header: `boost/asio/ssl/context.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::context::add_verify_path

Add a directory containing certificate authority files to be used for performing verification.

```
void add_verify_path(
    const std::string & path);
  » more...

boost::system::error_code add_verify_path(
    const std::string & path,
    boost::system::error_code & ec);
  » more...
```

ssl::context::add_verify_path (1 of 2 overloads)

Add a directory containing certificate authority files to be used for performing verification.

```
void add_verify_path(
    const std::string & path);
```

This function is used to specify the name of a directory containing certification authority certificates. Each file in the directory must contain a single certificate. The files must be named using the subject name's hash and an extension of ".0".

Parameters

path The name of a directory containing the certificates.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

Calls `SSL_CTX_load_verify_locations`.

ssl::context::add_verify_path (2 of 2 overloads)

Add a directory containing certificate authority files to be used for performing verification.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code add_verify_path(
    const std::string & path,
    boost::system::error_code & ec);
```

This function is used to specify the name of a directory containing certification authority certificates. Each file in the directory must contain a single certificate. The files must be named using the subject name's hash and an extension of ".0".

Parameters

path The name of a directory containing the certificates.

ec Set to indicate what error occurred, if any.

Remarks

Calls `SSL_CTX_load_verify_locations`.

ssl::context::context

Constructor.

```
explicit context(
    method m);
  » more...
```

Deprecated constructor taking a reference to an `io_service` object.

```
context(
    boost::asio::io_service & ,
    method m);
  » more...
```

Move-construct a context from another.

```
context(
    context && other);
  » more...
```

ssl::context::context (1 of 3 overloads)

Constructor.

```
context(
    method m);
```

ssl::context::context (2 of 3 overloads)

Deprecated constructor taking a reference to an `io_service` object.

```
context(
    boost::asio::io_service & ,
    method m);
```

ssl::context::context (3 of 3 overloads)

Move-construct a context from another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
context(
    context && other);
```

This constructor moves an SSL context from one object to another.

Parameters

other The other context object from which the move will occur.

Remarks

Following the move, the following operations only are valid for the moved-from object: * Destruction.

- As a target for move-assignment.

ssl::context::default_workarounds

Implement various bug workarounds.

```
static const int default_workarounds = implementation_defined;
```

ssl::context::file_format

File format types.

```
enum file_format
```

Values

asn1 ASN.1 file.

pem PEM file.

ssl::context::impl

(Deprecated: Use `native_handle()`.) Get the underlying implementation in the native type.

```
impl_type impl();
```

This function may be used to obtain the underlying implementation of the context. This is intended to allow access to context functionality that is not otherwise provided.

ssl::context::impl_type

(Deprecated: Use native_handle_type.) The native type of the SSL context.

```
typedef SSL_CTX * impl_type;
```

Requirements

Header: `boost/asio/ssl/context.hpp`

Convenience header: `boost/asio/ssl.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ssl::context::load_verify_file

Load a certification authority file for performing verification.

```
void load_verify_file(
    const std::string & filename);
  » more...

boost::system::error_code load_verify_file(
    const std::string & filename,
    boost::system::error_code & ec);
  » more...
```

ssl::context::load_verify_file (1 of 2 overloads)

Load a certification authority file for performing verification.

```
void load_verify_file(
    const std::string & filename);
```

This function is used to load one or more trusted certification authorities from a file.

Parameters

filename The name of a file containing certification authority certificates in PEM format.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

Calls `SSL_CTX_load_verify_locations`.

ssl::context::load_verify_file (2 of 2 overloads)

Load a certification authority file for performing verification.

```
boost::system::error_code load_verify_file(
    const std::string & filename,
    boost::system::error_code & ec);
```

This function is used to load the certificates for one or more trusted certification authorities from a file.

Parameters

filename The name of a file containing certification authority certificates in PEM format.

ec Set to indicate what error occurred, if any.

Remarks

Calls `SSL_CTX_load_verify_locations`.

ssl::context::method

Different methods supported by a context.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
enum method
```

Values

sslv2	Generic SSL version 2.
sslv2_client	SSL version 2 client.
sslv2_server	SSL version 2 server.
sslv3	Generic SSL version 3.
sslv3_client	SSL version 3 client.
sslv3_server	SSL version 3 server.
tlsv1	Generic TLS version 1.
tlsv1_client	TLS version 1 client.
tlsv1_server	TLS version 1 server.
sslv23	Generic SSL/TLS.
sslv23_client	SSL/TLS client.
sslv23_server	SSL/TLS server.

ssl::context::native_handle

Get the underlying implementation in the native type.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying implementation of the context. This is intended to allow access to context functionality that is not otherwise provided.

ssl::context::native_handle_type

The native handle type of the SSL context.

```
typedef SSL_CTX * native_handle_type;
```

Requirements

Header: `boost/asio/ssl/context.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::context::no_sslv2

Disable SSL v2.

```
static const int no_sslv2 = implementation_defined;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ssl::context::no_sslv3

Disable SSL v3.

```
static const int no_sslv3 = implementation_defined;
```

ssl::context::no_tlsv1

Disable TLS v1.

```
static const int no_tlsv1 = implementation_defined;
```

ssl::context::operator=

Move-assign a context from another.

```
context & operator=(
    context && other);
```

This assignment operator moves an SSL context from one object to another.

Parameters

other The other context object from which the move will occur.

Remarks

Following the move, the following operations only are valid for the moved-from object: * Destruction.

- As a target for move-assignment.

ssl::context::options

Bitmask type for SSL options.

```
typedef int options;
```

Requirements

Header: `boost/asio/ssl/context.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::context::password_purpose

Purpose of PEM password.

```
enum password_purpose
```

Values

for_reading The password is needed for reading/decryption.

for_writing The password is needed for writing/encryption.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ssl::context::set_default_verify_paths

Configures the context to use the default directories for finding certification authority certificates.

```
void set_default_verify_paths();
  » more...

boost::system::error_code set_default_verify_paths(
    boost::system::error_code & ec);
  » more...
```

ssl::context::set_default_verify_paths (1 of 2 overloads)

Configures the context to use the default directories for finding certification authority certificates.

```
void set_default_verify_paths();
```

This function specifies that the context should use the default, system-dependent directories for locating certification authority certificates.

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

Calls `SSL_CTX_set_default_verify_paths`.

ssl::context::set_default_verify_paths (2 of 2 overloads)

Configures the context to use the default directories for finding certification authority certificates.

```
boost::system::error_code set_default_verify_paths(
    boost::system::error_code & ec);
```

This function specifies that the context should use the default, system-dependent directories for locating certification authority certificates.

Parameters

ec	Set to indicate what error occurred, if any.

Remarks

Calls `SSL_CTX_set_default_verify_paths`.

ssl::context::set_options

Set options on the context.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void set_options(
    options o);
  » more...

boost::system::error_code set_options(
    options o,
    boost::system::error_code & ec);
  » more...
```

ssl::context::set_options (1 of 2 overloads)

Set options on the context.

```
void set_options(
    options o);
```

This function may be used to configure the SSL options used by the context.

Parameters

o A bitmask of options. The available option values are defined in the `ssl::context_base` class. The options are bitwise-ored with any existing value for the options.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

Calls `SSL_CTX_set_options`.

ssl::context::set_options (2 of 2 overloads)

Set options on the context.

```
boost::system::error_code set_options(
    options o,
    boost::system::error_code & ec);
```

This function may be used to configure the SSL options used by the context.

Parameters

o A bitmask of options. The available option values are defined in the `ssl::context_base` class. The options are bitwise-ored with any existing value for the options.

ec Set to indicate what error occurred, if any.

Remarks

Calls `SSL_CTX_set_options`.

ssl::context::set_password_callback

Set the password callback.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename PasswordCallback>
void set_password_callback(
    PasswordCallback callback);
  » more...

template<
    typename PasswordCallback>
boost::system::error_code set_password_callback(
    PasswordCallback callback,
    boost::system::error_code & ec);
  » more...
```

ssl::context::set_password_callback (1 of 2 overloads)

Set the password callback.

```
template<
    typename PasswordCallback>
void set_password_callback(
    PasswordCallback callback);
```

This function is used to specify a callback function to obtain password information about an encrypted key in PEM format.

Parameters

callback The function object to be used for obtaining the password. The function signature of the handler must be:

```
std::string password_callback(
  std::size_t max_length,  // The maximum size for a password.
  password_purpose purpose // Whether password is for reading or writing.
);
```

The return value of the callback is a string containing the password.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

Calls `SSL_CTX_set_default_passwd_cb`.

ssl::context::set_password_callback (2 of 2 overloads)

Set the password callback.

```
template<
    typename PasswordCallback>
boost::system::error_code set_password_callback(
    PasswordCallback callback,
    boost::system::error_code & ec);
```

This function is used to specify a callback function to obtain password information about an encrypted key in PEM format.

Parameters

callback The function object to be used for obtaining the password. The function signature of the handler must be:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::string password_callback(
  std::size_t max_length,  // The maximum size for a password.
  password_purpose purpose // Whether password is for reading or writing.
);
```

The return value of the callback is a string containing the password.

ec Set to indicate what error occurred, if any.

Remarks

Calls `SSL_CTX_set_default_passwd_cb`.

ssl::context::set_verify_callback

Set the callback used to verify peer certificates.

```
template<
    typename VerifyCallback>
void set_verify_callback(
    VerifyCallback callback);
  » more...

template<
    typename VerifyCallback>
boost::system::error_code set_verify_callback(
    VerifyCallback callback,
    boost::system::error_code & ec);
  » more...
```

ssl::context::set_verify_callback (1 of 2 overloads)

Set the callback used to verify peer certificates.

```
template<
    typename VerifyCallback>
void set_verify_callback(
    VerifyCallback callback);
```

This function is used to specify a callback function that will be called by the implementation when it needs to verify a peer certificate.

Parameters

callback The function object to be used for verifying a certificate. The function signature of the handler must be:

```
bool verify_callback(
  bool preverified, // True if the certificate passed pre-verification.
  verify_context& ctx // The peer certificate and other context.
);
```

The return value of the callback is true if the certificate has passed verification, false otherwise.

Exceptions

boost::system::system_error Thrown on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

Calls `SSL_CTX_set_verify`.

ssl::context::set_verify_callback (2 of 2 overloads)

Set the callback used to verify peer certificates.

```
template<
    typename VerifyCallback>
boost::system::error_code set_verify_callback(
    VerifyCallback callback,
    boost::system::error_code & ec);
```

This function is used to specify a callback function that will be called by the implementation when it needs to verify a peer certificate.

Parameters

callback — The function object to be used for verifying a certificate. The function signature of the handler must be:

```
bool verify_callback(
  bool preverified, // True if the certificate passed pre-verification.
  verify_context& ctx // The peer certificate and other context.
);
```

The return value of the callback is true if the certificate has passed verification, false otherwise.

ec — Set to indicate what error occurred, if any.

Remarks

Calls `SSL_CTX_set_verify`.

ssl::context::set_verify_mode

Set the peer verification mode.

```
void set_verify_mode(
    verify_mode v);
  » more...

boost::system::error_code set_verify_mode(
    verify_mode v,
    boost::system::error_code & ec);
  » more...
```

ssl::context::set_verify_mode (1 of 2 overloads)

Set the peer verification mode.

```
void set_verify_mode(
    verify_mode v);
```

This function may be used to configure the peer verification mode used by the context.

Parameters

v — A bitmask of peer verification modes. See `ssl::verify_mode` for available values.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

Calls `SSL_CTX_set_verify`.

ssl::context::set_verify_mode (2 of 2 overloads)

Set the peer verification mode.

```
boost::system::error_code set_verify_mode(
    verify_mode v,
    boost::system::error_code & ec);
```

This function may be used to configure the peer verification mode used by the context.

Parameters

v	A bitmask of peer verification modes. See `ssl::verify_mode` for available values.
ec	Set to indicate what error occurred, if any.

Remarks

Calls `SSL_CTX_set_verify`.

ssl::context::single_dh_use

Always create a new key when using tmp_dh parameters.

```
static const int single_dh_use = implementation_defined;
```

ssl::context::use_certificate_chain_file

Use a certificate chain from a file.

```
void use_certificate_chain_file(
    const std::string & filename);
  » more...

boost::system::error_code use_certificate_chain_file(
    const std::string & filename,
    boost::system::error_code & ec);
  » more...
```

ssl::context::use_certificate_chain_file (1 of 2 overloads)

Use a certificate chain from a file.

```
void use_certificate_chain_file(
    const std::string & filename);
```

This function is used to load a certificate chain into the context from a file.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

filename The name of the file containing the certificate. The file must use the PEM format.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

Calls `SSL_CTX_use_certificate_chain_file`.

ssl::context::use_certificate_chain_file (2 of 2 overloads)

Use a certificate chain from a file.

```
boost::system::error_code use_certificate_chain_file(
    const std::string & filename,
    boost::system::error_code & ec);
```

This function is used to load a certificate chain into the context from a file.

Parameters

filename The name of the file containing the certificate. The file must use the PEM format.

ec Set to indicate what error occurred, if any.

Remarks

Calls `SSL_CTX_use_certificate_chain_file`.

ssl::context::use_certificate_file

Use a certificate from a file.

```
void use_certificate_file(
    const std::string & filename,
    file_format format);
  » more...

boost::system::error_code use_certificate_file(
    const std::string & filename,
    file_format format,
    boost::system::error_code & ec);
  » more...
```

ssl::context::use_certificate_file (1 of 2 overloads)

Use a certificate from a file.

```
void use_certificate_file(
    const std::string & filename,
    file_format format);
```

This function is used to load a certificate into the context from a file.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

filename	The name of the file containing the certificate.
format	The file format (ASN.1 or PEM).

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

Calls `SSL_CTX_use_certificate_file`.

ssl::context::use_certificate_file (2 of 2 overloads)

Use a certificate from a file.

```
boost::system::error_code use_certificate_file(
    const std::string & filename,
    file_format format,
    boost::system::error_code & ec);
```

This function is used to load a certificate into the context from a file.

Parameters

filename	The name of the file containing the certificate.
format	The file format (ASN.1 or PEM).
ec	Set to indicate what error occurred, if any.

Remarks

Calls `SSL_CTX_use_certificate_file`.

ssl::context::use_private_key_file

Use a private key from a file.

```
void use_private_key_file(
    const std::string & filename,
    file_format format);
  » more...

boost::system::error_code use_private_key_file(
    const std::string & filename,
    file_format format,
    boost::system::error_code & ec);
  » more...
```

ssl::context::use_private_key_file (1 of 2 overloads)

Use a private key from a file.

```
void use_private_key_file(
    const std::string & filename,
    file_format format);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This function is used to load a private key into the context from a file.

Parameters

filename	The name of the file containing the private key.
format	The file format (ASN.1 or PEM).

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

Calls `SSL_CTX_use_PrivateKey_file`.

ssl::context::use_private_key_file (2 of 2 overloads)

Use a private key from a file.

```
boost::system::error_code use_private_key_file(
    const std::string & filename,
    file_format format,
    boost::system::error_code & ec);
```

This function is used to load a private key into the context from a file.

Parameters

filename	The name of the file containing the private key.
format	The file format (ASN.1 or PEM).
ec	Set to indicate what error occurred, if any.

Remarks

Calls `SSL_CTX_use_PrivateKey_file`.

ssl::context::use_rsa_private_key_file

Use an RSA private key from a file.

```
void use_rsa_private_key_file(
    const std::string & filename,
    file_format format);
  » more...

boost::system::error_code use_rsa_private_key_file(
    const std::string & filename,
    file_format format,
    boost::system::error_code & ec);
  » more...
```

ssl::context::use_rsa_private_key_file (1 of 2 overloads)

Use an RSA private key from a file.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void use_rsa_private_key_file(
    const std::string & filename,
    file_format format);
```

This function is used to load an RSA private key into the context from a file.

Parameters

filename The name of the file containing the RSA private key.

format The file format (ASN.1 or PEM).

Exceptions

boost::system::system_error Thrown on failure.

Remarks

Calls `SSL_CTX_use_RSAPrivateKey_file`.

ssl::context::use_rsa_private_key_file (2 of 2 overloads)

Use an RSA private key from a file.

```
boost::system::error_code use_rsa_private_key_file(
    const std::string & filename,
    file_format format,
    boost::system::error_code & ec);
```

This function is used to load an RSA private key into the context from a file.

Parameters

filename The name of the file containing the RSA private key.

format The file format (ASN.1 or PEM).

ec Set to indicate what error occurred, if any.

Remarks

Calls `SSL_CTX_use_RSAPrivateKey_file`.

ssl::context::use_tmp_dh_file

Use the specified file to obtain the temporary Diffie-Hellman parameters.

```
void use_tmp_dh_file(
    const std::string & filename);
  » more...

boost::system::error_code use_tmp_dh_file(
    const std::string & filename,
    boost::system::error_code & ec);
  » more...
```

ssl::context::use_tmp_dh_file (1 of 2 overloads)

Use the specified file to obtain the temporary Diffie-Hellman parameters.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void use_tmp_dh_file(
    const std::string & filename);
```

This function is used to load Diffie-Hellman parameters into the context from a file.

Parameters

filename The name of the file containing the Diffie-Hellman parameters. The file must use the PEM format.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

Calls `SSL_CTX_set_tmp_dh`.

ssl::context::use_tmp_dh_file (2 of 2 overloads)

Use the specified file to obtain the temporary Diffie-Hellman parameters.

```
boost::system::error_code use_tmp_dh_file(
    const std::string & filename,
    boost::system::error_code & ec);
```

This function is used to load Diffie-Hellman parameters into the context from a file.

Parameters

filename The name of the file containing the Diffie-Hellman parameters. The file must use the PEM format.

ec Set to indicate what error occurred, if any.

Remarks

Calls `SSL_CTX_set_tmp_dh`.

ssl::context::~context

Destructor.

```
~context();
```

ssl::context_base

The `ssl::context_base` class is used as a base for the basic_context class template so that we have a common place to define various enums.

```
class context_base
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
file_format	File format types.
method	Different methods supported by a context.
options	Bitmask type for SSL options.
password_purpose	Purpose of PEM password.

Protected Member Functions

Name	Description
~context_base	Protected destructor to prevent deletion through this type.

Data Members

Name	Description
default_workarounds	Implement various bug workarounds.
no_sslv2	Disable SSL v2.
no_sslv3	Disable SSL v3.
no_tlsv1	Disable TLS v1.
single_dh_use	Always create a new key when using tmp_dh parameters.

Requirements

Header: `boost/asio/ssl/context_base.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::context_base::default_workarounds

Implement various bug workarounds.

```
static const int default_workarounds = implementation_defined;
```

ssl::context_base::file_format

File format types.

```
enum file_format
```

Values

asn1 ASN.1 file.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

pem PEM file.

ssl::context_base::method

Different methods supported by a context.

```
enum method
```

Values

sslv2	Generic SSL version 2.
sslv2_client	SSL version 2 client.
sslv2_server	SSL version 2 server.
sslv3	Generic SSL version 3.
sslv3_client	SSL version 3 client.
sslv3_server	SSL version 3 server.
tlsv1	Generic TLS version 1.
tlsv1_client	TLS version 1 client.
tlsv1_server	TLS version 1 server.
sslv23	Generic SSL/TLS.
sslv23_client	SSL/TLS client.
sslv23_server	SSL/TLS server.

ssl::context_base::no_sslv2

Disable SSL v2.

```
static const int no_sslv2 = implementation_defined;
```

ssl::context_base::no_sslv3

Disable SSL v3.

```
static const int no_sslv3 = implementation_defined;
```

ssl::context_base::no_tlsv1

Disable TLS v1.

```
static const int no_tlsv1 = implementation_defined;
```

ssl::context_base::options

Bitmask type for SSL options.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef int options;
```

Requirements

Header: `boost/asio/ssl/context_base.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::context_base::password_purpose

Purpose of PEM password.

```
enum password_purpose
```

Values

for_reading	The password is needed for reading/decryption.
for_writing	The password is needed for writing/encryption.

ssl::context_base::single_dh_use

Always create a new key when using tmp_dh parameters.

```
static const int single_dh_use = implementation_defined;
```

ssl::context_base::~context_base

Protected destructor to prevent deletion through this type.

```
~context_base();
```

ssl::rfc2818_verification

Verifies a certificate against a hostname according to the rules described in RFC 2818.

```
class rfc2818_verification
```

Types

Name	Description
result_type	The type of the function object's result.

Member Functions

Name	Description
operator()	Perform certificate verification.
rfc2818_verification	Constructor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

The following example shows how to synchronously open a secure connection to a given host name:

```
using boost::asio::ip::tcp;
namespace ssl = boost::asio::ssl;
typedef ssl::stream<tcp::socket> ssl_socket;

// Create a context that uses the default paths for finding CA certificates.
ssl::context ctx(ssl::context::sslv23);
ctx.set_default_verify_paths();

// Open a socket and connect it to the remote host.
boost::asio::io_service io_service;
ssl_socket sock(io_service, ctx);
tcp::resolver resolver(io_service);
tcp::resolver::query query("host.name", "https");
boost::asio::connect(sock.lowest_layer(), resolver.resolve(query));
sock.lowest_layer().set_option(tcp::no_delay(true));

// Perform SSL handshake and verify the remote host's certificate.
sock.set_verify_mode(ssl::verify_peer);
sock.set_verify_callback(ssl::rfc2818_verification("host.name"));
sock.handshake(ssl_socket::client);

// ... read and write as normal ...
```

Requirements

Header: `boost/asio/ssl/rfc2818_verification.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::rfc2818_verification::operator()

Perform certificate verification.

```
bool operator()(
    bool preverified,
    verify_context & ctx) const;
```

ssl::rfc2818_verification::result_type

The type of the function object's result.

```
typedef bool result_type;
```

Requirements

Header: `boost/asio/ssl/rfc2818_verification.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::rfc2818_verification::rfc2818_verification

Constructor.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
rfc2818_verification(
    const std::string & host);
```

ssl::stream

Provides stream-oriented functionality using SSL.

```
template<
    typename Stream>
class stream :
  public ssl::stream_base,
  noncopyable
```

Types

Name	Description
impl_struct	Structure for use with deprecated impl_type.
handshake_type	Different handshake types.
impl_type	(Deprecated: Use native_handle_type.) The underlying implementation type.
lowest_layer_type	The type of the lowest layer.
native_handle_type	The native handle type of the SSL stream.
next_layer_type	The type of the next layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
async_handshake	Start an asynchronous SSL handshake.
async_read_some	Start an asynchronous read.
async_shutdown	Asynchronously shut down SSL on the stream.
async_write_some	Start an asynchronous write.
get_io_service	Get the io_service associated with the object.
handshake	Perform SSL handshaking.
impl	(Deprecated: Use native_handle().) Get the underlying implementation in the native type.
lowest_layer	Get a reference to the lowest layer.
native_handle	Get the underlying implementation in the native type.
next_layer	Get a reference to the next layer.
read_some	Read some data from the stream.
set_verify_callback	Set the callback used to verify peer certificates.
set_verify_mode	Set the peer verification mode.
shutdown	Shut down SSL on the stream.
stream	Construct a stream.
write_some	Write some data to the stream.
~stream	Destructor.

The stream class template provides asynchronous and blocking stream-oriented functionality using SSL.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe. The application must also ensure that all asynchronous operations are performed within the same implicit or explicit strand.

Example

To use the SSL stream template with an `ip::tcp::socket`, you would write:

```
boost::asio::io_service io_service;
boost::asio::ssl::context ctx(boost::asio::ssl::context::sslv23);
boost::asio::ssl::stream<asio:ip::tcp::socket> sock(io_service, ctx);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/ssl/stream.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::stream::async_handshake

Start an asynchronous SSL handshake.

```
template<
    typename HandshakeHandler>
void async_handshake(
    handshake_type type,
    HandshakeHandler handler);
```

This function is used to asynchronously perform an SSL handshake on the stream. This function call always returns immediately.

Parameters

type — The type of handshaking to be performed, i.e. as a client or as a server.

handler — The handler to be called when the handshake operation completes. Copies will be made of the handler as required. The equivalent function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error // Result of operation.
);
```

ssl::stream::async_read_some

Start an asynchronous read.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_some(
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

This function is used to asynchronously read one or more bytes of data from the stream. The function call always returns immediately.

Parameters

buffers — The buffers into which the data will be read. Although the buffers object may be copied as necessary, ownership of the underlying buffers is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the read operation completes. Copies will be made of the handler as required. The equivalent function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes read.
);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

The async_read_some operation may not read all of the requested number of bytes. Consider using the `async_read` function if you need to ensure that the requested amount of data is read before the asynchronous operation completes.

ssl::stream::async_shutdown

Asynchronously shut down SSL on the stream.

```
template<
    typename ShutdownHandler>
void async_shutdown(
    ShutdownHandler handler);
```

This function is used to asynchronously shut down SSL on the stream. This function call always returns immediately.

Parameters

handler — The handler to be called when the handshake operation completes. Copies will be made of the handler as required. The equivalent function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error // Result of operation.
);
```

ssl::stream::async_write_some

Start an asynchronous write.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_some(
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

This function is used to asynchronously write one or more bytes of data to the stream. The function call always returns immediately.

Parameters

buffers — The data to be written to the stream. Although the buffers object may be copied as necessary, ownership of the underlying buffers is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the write operation completes. Copies will be made of the handler as required. The equivalent function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes written.
);
```

Remarks

The async_write_some operation may not transmit all of the data to the peer. Consider using the `async_write` function if you need to ensure that all data is written before the blocking operation completes.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ssl::stream::get_io_service

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the stream uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that stream will use to dispatch handlers. Ownership is not transferred to the caller.

ssl::stream::handshake

Perform SSL handshaking.

```
void handshake(
    handshake_type type);
  » more...

boost::system::error_code handshake(
    handshake_type type,
    boost::system::error_code & ec);
  » more...
```

ssl::stream::handshake (1 of 2 overloads)

Perform SSL handshaking.

```
void handshake(
    handshake_type type);
```

This function is used to perform SSL handshaking on the stream. The function call will block until handshaking is complete or an error occurs.

Parameters

type The type of handshaking to be performed, i.e. as a client or as a server.

Exceptions

boost::system::system_error Thrown on failure.

ssl::stream::handshake (2 of 2 overloads)

Perform SSL handshaking.

```
boost::system::error_code handshake(
    handshake_type type,
    boost::system::error_code & ec);
```

This function is used to perform SSL handshaking on the stream. The function call will block until handshaking is complete or an error occurs.

Parameters

type The type of handshaking to be performed, i.e. as a client or as a server.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ec Set to indicate what error occurred, if any.

ssl::stream::handshake_type

Different handshake types.

```
enum handshake_type
```

Values

client Perform handshaking as a client.

server Perform handshaking as a server.

ssl::stream::impl

(Deprecated: Use `native_handle()`.) Get the underlying implementation in the native type.

```
impl_type impl();
```

This function may be used to obtain the underlying implementation of the context. This is intended to allow access to stream functionality that is not otherwise provided.

ssl::stream::impl_type

(Deprecated: Use native_handle_type.) The underlying implementation type.

```
typedef impl_struct * impl_type;
```

Requirements

Header: `boost/asio/ssl/stream.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::stream::lowest_layer

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
  » more...

const lowest_layer_type & lowest_layer() const;
  » more...
```

ssl::stream::lowest_layer (1 of 2 overloads)

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

This function returns a reference to the lowest layer in a stack of stream layers.

Return Value

A reference to the lowest layer in the stack of stream layers. Ownership is not transferred to the caller.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ssl::stream::lowest_layer (2 of 2 overloads)

Get a reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

This function returns a reference to the lowest layer in a stack of stream layers.

Return Value

A reference to the lowest layer in the stack of stream layers. Ownership is not transferred to the caller.

ssl::stream::lowest_layer_type

The type of the lowest layer.

```
typedef next_layer_type::lowest_layer_type lowest_layer_type;
```

Requirements

Header: `boost/asio/ssl/stream.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::stream::native_handle

Get the underlying implementation in the native type.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying implementation of the context. This is intended to allow access to context functionality that is not otherwise provided.

Example

The `native_handle()` function returns a pointer of type `SSL*` that is suitable for passing to functions such as `SSL_get_verify_result` and `SSL_get_peer_certificate`:

```
boost::asio::ssl::stream<asio:ip::tcp::socket> sock(io_service, ctx);

// ... establish connection and perform handshake ...

if (X509* cert = SSL_get_peer_certificate(sock.native_handle()))
{
  if (SSL_get_verify_result(sock.native_handle()) == X509_V_OK)
  {
    // ...
  }
}
```

ssl::stream::native_handle_type

The native handle type of the SSL stream.

```
typedef SSL * native_handle_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/ssl/stream.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::stream::next_layer

Get a reference to the next layer.

```
const next_layer_type & next_layer() const;
  » more...

next_layer_type & next_layer();
  » more...
```

ssl::stream::next_layer (1 of 2 overloads)

Get a reference to the next layer.

```
const next_layer_type & next_layer() const;
```

This function returns a reference to the next layer in a stack of stream layers.

Return Value

A reference to the next layer in the stack of stream layers. Ownership is not transferred to the caller.

ssl::stream::next_layer (2 of 2 overloads)

Get a reference to the next layer.

```
next_layer_type & next_layer();
```

This function returns a reference to the next layer in a stack of stream layers.

Return Value

A reference to the next layer in the stack of stream layers. Ownership is not transferred to the caller.

ssl::stream::next_layer_type

The type of the next layer.

```
typedef boost::remove_reference< Stream >::type next_layer_type;
```

Requirements

Header: `boost/asio/ssl/stream.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::stream::read_some

Read some data from the stream.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
  » more...

template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

ssl::stream::read_some (1 of 2 overloads)

Read some data from the stream.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
```

This function is used to read data from the stream. The function call will block until one or more bytes of data has been read successfully, or until an error occurs.

Parameters

buffers The buffers into which the data will be read.

Return Value

The number of bytes read.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

The read_some operation may not read all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

ssl::stream::read_some (2 of 2 overloads)

Read some data from the stream.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to read data from the stream. The function call will block until one or more bytes of data has been read successfully, or until an error occurs.

Parameters

buffers The buffers into which the data will be read.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes read. Returns 0 if an error occurred.

Remarks

The read_some operation may not read all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

ssl::stream::set_verify_callback

Set the callback used to verify peer certificates.

```
template<
    typename VerifyCallback>
void set_verify_callback(
    VerifyCallback callback);
  » more...

template<
    typename VerifyCallback>
boost::system::error_code set_verify_callback(
    VerifyCallback callback,
    boost::system::error_code & ec);
  » more...
```

ssl::stream::set_verify_callback (1 of 2 overloads)

Set the callback used to verify peer certificates.

```
template<
    typename VerifyCallback>
void set_verify_callback(
    VerifyCallback callback);
```

This function is used to specify a callback function that will be called by the implementation when it needs to verify a peer certificate.

Parameters

callback The function object to be used for verifying a certificate. The function signature of the handler must be:

```
bool verify_callback(
  bool preverified, // True if the certificate passed pre-verification.
  verify_context& ctx // The peer certificate and other context.
);
```

The return value of the callback is true if the certificate has passed verification, false otherwise.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

Calls `SSL_set_verify`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ssl::stream::set_verify_callback (2 of 2 overloads)

Set the callback used to verify peer certificates.

```
template<
    typename VerifyCallback>
boost::system::error_code set_verify_callback(
    VerifyCallback callback,
    boost::system::error_code & ec);
```

This function is used to specify a callback function that will be called by the implementation when it needs to verify a peer certificate.

Parameters

callback The function object to be used for verifying a certificate. The function signature of the handler must be:

```
bool verify_callback(
  bool preverified, // True if the certificate passed pre-verification.
  verify_context& ctx // The peer certificate and other context.
);
```

The return value of the callback is true if the certificate has passed verification, false otherwise.

ec Set to indicate what error occurred, if any.

Remarks

Calls `SSL_set_verify`.

ssl::stream::set_verify_mode

Set the peer verification mode.

```
void set_verify_mode(
    verify_mode v);
  » more...

boost::system::error_code set_verify_mode(
    verify_mode v,
    boost::system::error_code & ec);
  » more...
```

ssl::stream::set_verify_mode (1 of 2 overloads)

Set the peer verification mode.

```
void set_verify_mode(
    verify_mode v);
```

This function may be used to configure the peer verification mode used by the stream. The new mode will override the mode inherited from the context.

Parameters

v A bitmask of peer verification modes. See `ssl::verify_mode` for available values.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Remarks

Calls `SSL_set_verify`.

ssl::stream::set_verify_mode (2 of 2 overloads)

Set the peer verification mode.

```
boost::system::error_code set_verify_mode(
    verify_mode v,
    boost::system::error_code & ec);
```

This function may be used to configure the peer verification mode used by the stream. The new mode will override the mode inherited from the context.

Parameters

v	A bitmask of peer verification modes. See `ssl::verify_mode` for available values.
ec	Set to indicate what error occurred, if any.

Remarks

Calls `SSL_set_verify`.

ssl::stream::shutdown

Shut down SSL on the stream.

```
void shutdown();
  » more...

boost::system::error_code shutdown(
    boost::system::error_code & ec);
  » more...
```

ssl::stream::shutdown (1 of 2 overloads)

Shut down SSL on the stream.

```
void shutdown();
```

This function is used to shut down SSL on the stream. The function call will block until SSL has been shut down or an error occurs.

Exceptions

boost::system::system_error	Thrown on failure.

ssl::stream::shutdown (2 of 2 overloads)

Shut down SSL on the stream.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code shutdown(
    boost::system::error_code & ec);
```

This function is used to shut down SSL on the stream. The function call will block until SSL has been shut down or an error occurs.

Parameters

ec Set to indicate what error occurred, if any.

ssl::stream::stream

Construct a stream.

```
template<
    typename Arg>
stream(
    Arg & arg,
    context & ctx);
```

This constructor creates a stream and initialises the underlying stream object.

Parameters

arg The argument to be passed to initialise the underlying stream.

ctx The SSL context to be used for the stream.

ssl::stream::write_some

Write some data to the stream.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
  » more...

template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

ssl::stream::write_some (1 of 2 overloads)

Write some data to the stream.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
```

This function is used to write data on the stream. The function call will block until one or more bytes of data has been written successfully, or until an error occurs.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

buffers The data to be written.

Return Value

The number of bytes written.

Exceptions

boost::system::system_error Thrown on failure.

Remarks

The write_some operation may not transmit all of the data to the peer. Consider using the `write` function if you need to ensure that all data is written before the blocking operation completes.

ssl::stream::write_some (2 of 2 overloads)

Write some data to the stream.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to write data on the stream. The function call will block until one or more bytes of data has been written successfully, or until an error occurs.

Parameters

buffers The data to be written to the stream.

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes written. Returns 0 if an error occurred.

Remarks

The write_some operation may not transmit all of the data to the peer. Consider using the `write` function if you need to ensure that all data is written before the blocking operation completes.

ssl::stream::~stream

Destructor.

```
~stream();
```

ssl::stream::impl_struct

Structure for use with deprecated impl_type.

```
struct impl_struct
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Data Members

Name	Description
ssl	

Requirements

Header: `boost/asio/ssl/stream.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::stream::impl_struct::ssl

```
SSL * ssl;
```

ssl::stream_base

The `ssl::stream_base` class is used as a base for the `ssl::stream` class template so that we have a common place to define various enums.

```
class stream_base
```

Types

Name	Description
handshake_type	Different handshake types.

Protected Member Functions

Name	Description
~stream_base	Protected destructor to prevent deletion through this type.

Requirements

Header: `boost/asio/ssl/stream_base.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::stream_base::handshake_type

Different handshake types.

```
enum handshake_type
```

Values

client — Perform handshaking as a client.

server — Perform handshaking as a server.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ssl::stream_base::~stream_base

Protected destructor to prevent deletion through this type.

```
~stream_base();
```

ssl::verify_client_once

Do not request client certificate on renegotiation. Ignored unless `ssl::verify_peer` is set.

```
const int verify_client_once = implementation_defined;
```

Requirements

Header: `boost/asio/ssl/verify_mode.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::verify_context

A simple wrapper around the X509_STORE_CTX type, used during verification of a peer certificate.

```
class verify_context :
  noncopyable
```

Types

Name	Description
native_handle_type	The native handle type of the verification context.

Member Functions

Name	Description
native_handle	Get the underlying implementation in the native type.
verify_context	Constructor.

Remarks

The `ssl::verify_context` does not own the underlying X509_STORE_CTX object.

Requirements

Header: `boost/asio/ssl/verify_context.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::verify_context::native_handle

Get the underlying implementation in the native type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying implementation of the context. This is intended to allow access to context functionality that is not otherwise provided.

ssl::verify_context::native_handle_type

The native handle type of the verification context.

```
typedef X509_STORE_CTX * native_handle_type;
```

Requirements

Header: `boost/asio/ssl/verify_context.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::verify_context::verify_context

Constructor.

```
verify_context(
    native_handle_type handle);
```

ssl::verify_fail_if_no_peer_cert

Fail verification if the peer has no certificate. Ignored unless `ssl::verify_peer` is set.

```
const int verify_fail_if_no_peer_cert = implementation_defined;
```

Requirements

Header: `boost/asio/ssl/verify_mode.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::verify_mode

Bitmask type for peer verification.

```
typedef int verify_mode;
```

Possible values are:

- `ssl::verify_none`
- `ssl::verify_peer`
- `ssl::verify_fail_if_no_peer_cert`
- `ssl::verify_client_once`

Requirements

Header: `boost/asio/ssl/verify_mode.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Convenience header: `boost/asio/ssl.hpp`

ssl::verify_none

No verification.

```
const int verify_none = implementation_defined;
```

Requirements

Header: `boost/asio/ssl/verify_mode.hpp`

Convenience header: `boost/asio/ssl.hpp`

ssl::verify_peer

Verify the peer.

```
const int verify_peer = implementation_defined;
```

Requirements

Header: `boost/asio/ssl/verify_mode.hpp`

Convenience header: `boost/asio/ssl.hpp`

steady_timer

Typedef for a timer based on the steady clock.

```
typedef basic_waitable_timer< chrono::steady_clock > steady_timer;
```

Types

Name	Description
clock_type	The clock type.
duration	The duration type of the clock.
implementation_type	The underlying implementation type of I/O object.
service_type	The type of the service that will be used to provide I/O operations.
time_point	The time point type of the clock.
traits_type	The wait traits type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
async_wait	Start an asynchronous wait on the timer.
basic_waitable_timer	Constructor. Constructor to set a particular expiry time as an absolute time. Constructor to set a particular expiry time relative to now.
cancel	Cancel any asynchronous operations that are waiting on the timer.
cancel_one	Cancels one asynchronous operation that is waiting on the timer.
expires_at	Get the timer's expiry time as an absolute time. Set the timer's expiry time as an absolute time.
expires_from_now	Get the timer's expiry time relative to now. Set the timer's expiry time relative to now.
get_io_service	Get the io_service associated with the object.
wait	Perform a blocking wait on the timer.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_waitable_timer` class template provides the ability to perform a blocking or asynchronous wait for a timer to expire.

A waitable timer is always in one of two states: "expired" or "not expired". If the `wait()` or `async_wait()` function is called on an expired timer, the wait operation will complete immediately.

Most applications will use the boost::asio::waitable_timer typedef.

Remarks

This waitable timer functionality is for use with the C++11 standard library's `<chrono>` facility, or with the Boost.Chrono library.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Examples

Performing a blocking wait:

```
// Construct a timer without setting an expiry time.
boost::asio::waitable_timer timer(io_service);

// Set an expiry time relative to now.
timer.expires_from_now(boost::posix_time::seconds(5));

// Wait for the timer to expire.
timer.wait();
```

Performing an asynchronous wait:

```
void handler(const boost::system::error_code& error)
{
  if (!error)
  {
    // Timer expired.
  }
}

...

// Construct a timer with an absolute expiry time.
boost::asio::waitable_timer timer(io_service,
    boost::posix_time::time_from_string("2005-12-07 23:59:59.000"));

// Start an asynchronous wait.
timer.async_wait(handler);
```

Changing an active waitable_timer's expiry time

Changing the expiry time of a timer while there are pending asynchronous waits causes those wait operations to be cancelled. To ensure that the action associated with the timer is performed only once, use something like this: used:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void on_some_event()
{
  if (my_timer.expires_from_now(seconds(5)) > 0)
  {
    // We managed to cancel the timer. Start new asynchronous wait.
    my_timer.async_wait(on_timeout);
  }
  else
  {
    // Too late, timer has already expired!
  }
}

void on_timeout(const boost::system::error_code& e)
{
  if (e != boost::asio::error::operation_aborted)
  {
    // Timer was not cancelled, take necessary action.
  }
}
```

- The `boost::asio::basic_waitable_timer::expires_from_now()` function cancels any pending asynchronous waits, and returns the number of asynchronous waits that were cancelled. If it returns 0 then you were too late and the wait handler has already been executed, or will soon be executed. If it returns 1 then the wait handler was successfully cancelled.
- If a wait handler is cancelled, the boost::system::error_code passed to it contains the value `boost::asio::error::operation_aborted`.

This typedef uses the C++11 `<chrono>` standard library facility, if available. Otherwise, it may use the Boost.Chrono library. To explicitly utilise Boost.Chrono, use the `basic_waitable_timer` template directly:

```
typedef basic_waitable_timer<boost::chrono::steady_clock> timer;
```

Requirements

Header: `boost/asio/steady_timer.hpp`

Convenience header: None

strand

Typedef for backwards compatibility.

```
typedef boost::asio::io_service::strand strand;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
dispatch	Request the strand to invoke the given handler.
get_io_service	Get the io_service associated with the strand.
post	Request the strand to invoke the given handler and return immediately.
strand	Constructor.
wrap	Create a new handler that automatically dispatches the wrapped handler on the strand.
~strand	Destructor.

The `io_service::strand` class provides the ability to post and dispatch handlers with the guarantee that none of those handlers will execute concurrently.

Order of handler invocation

Given:

- a strand object `s`
- an object `a` meeting completion handler requirements
- an object `a1` which is an arbitrary copy of `a` made by the implementation
- an object `b` meeting completion handler requirements
- an object `b1` which is an arbitrary copy of `b` made by the implementation

if any of the following conditions are true:

- `s.post(a)` happens-before `s.post(b)`
- `s.post(a)` happens-before `s.dispatch(b)`, where the latter is performed outside the strand
- `s.dispatch(a)` happens-before `s.post(b)`, where the former is performed outside the strand
- `s.dispatch(a)` happens-before `s.dispatch(b)`, where both are performed outside the strand

then `asio_handler_invoke(a1, &a1)` happens-before `asio_handler_invoke(b1, &b1)`.

Note that in the following case:

```
async_op_1(..., s.wrap(a));
async_op_2(..., s.wrap(b));
```

the completion of the first async operation will perform `s.dispatch(a)`, and the second will perform `s.dispatch(b)`, but the order in which those are performed is unspecified. That is, you cannot state whether one happens-before the other. Therefore none of the above conditions are met and no ordering guarantee is made.

Thread Safety

Distinct objects: Safe.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Shared objects: Safe.

Requirements

Header: `boost/asio/strand.hpp`

Convenience header: `boost/asio.hpp`

stream_socket_service

Default service implementation for a stream socket.

```
template<
    typename Protocol>
class stream_socket_service :
  public io_service::service
```

Types

Name	Description
endpoint_type	The endpoint type.
implementation_type	The type of a stream socket implementation.
native_handle_type	The native socket type.
native_type	(Deprecated: Use native_handle_type.) The native socket type.
protocol_type	The protocol type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
assign	Assign an existing native socket to a stream socket.
async_connect	Start an asynchronous connect.
async_receive	Start an asynchronous receive.
async_send	Start an asynchronous send.
at_mark	Determine whether the socket is at the out-of-band data mark.
available	Determine the number of bytes available for reading.
bind	Bind the stream socket to the specified local endpoint.
cancel	Cancel all asynchronous operations associated with the socket.
close	Close a stream socket implementation.
connect	Connect the stream socket to the specified endpoint.
construct	Construct a new stream socket implementation.
destroy	Destroy a stream socket implementation.
get_io_service	Get the io_service object that owns the service.
get_option	Get a socket option.
io_control	Perform an IO control command on the socket.
is_open	Determine whether the socket is open.
local_endpoint	Get the local endpoint.
move_assign	Move-assign from another stream socket implementation.
move_construct	Move-construct a new stream socket implementation.
native	(Deprecated: Use native_handle().) Get the native socket implementation.
native_handle	Get the native socket implementation.
native_non_blocking	Gets the non-blocking mode of the native socket implementation. Sets the non-blocking mode of the native socket implementation.
non_blocking	Gets the non-blocking mode of the socket. Sets the non-blocking mode of the socket.
open	Open a stream socket.
receive	Receive some data from the peer.
remote_endpoint	Get the remote endpoint.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Name	Description
send	Send the given data to the peer.
set_option	Set a socket option.
shutdown	Disable sends or receives on the socket.
stream_socket_service	Construct a new stream socket service for the specified io_service.

Data Members

Name	Description
id	The unique service identifier.

Requirements

Header: `boost/asio/stream_socket_service.hpp`

Convenience header: `boost/asio.hpp`

stream_socket_service::assign

Assign an existing native socket to a stream socket.

```
boost::system::error_code assign(
    implementation_type & impl,
    const protocol_type & protocol,
    const native_handle_type & native_socket,
    boost::system::error_code & ec);
```

stream_socket_service::async_connect

Start an asynchronous connect.

```
template<
    typename ConnectHandler>
void async_connect(
    implementation_type & impl,
    const endpoint_type & peer_endpoint,
    ConnectHandler handler);
```

stream_socket_service::async_receive

Start an asynchronous receive.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_receive(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    ReadHandler handler);
```

stream_socket_service::async_send

Start an asynchronous send.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_send(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    WriteHandler handler);
```

stream_socket_service::at_mark

Determine whether the socket is at the out-of-band data mark.

```
bool at_mark(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

stream_socket_service::available

Determine the number of bytes available for reading.

```
std::size_t available(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

stream_socket_service::bind

Bind the stream socket to the specified local endpoint.

```
boost::system::error_code bind(
    implementation_type & impl,
    const endpoint_type & endpoint,
    boost::system::error_code & ec);
```

stream_socket_service::cancel

Cancel all asynchronous operations associated with the socket.

```
boost::system::error_code cancel(
    implementation_type & impl,
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

stream_socket_service::close

Close a stream socket implementation.

```
boost::system::error_code close(
    implementation_type & impl,
    boost::system::error_code & ec);
```

stream_socket_service::connect

Connect the stream socket to the specified endpoint.

```
boost::system::error_code connect(
    implementation_type & impl,
    const endpoint_type & peer_endpoint,
    boost::system::error_code & ec);
```

stream_socket_service::construct

Construct a new stream socket implementation.

```
void construct(
    implementation_type & impl);
```

stream_socket_service::destroy

Destroy a stream socket implementation.

```
void destroy(
    implementation_type & impl);
```

stream_socket_service::endpoint_type

The endpoint type.

```
typedef Protocol::endpoint endpoint_type;
```

Requirements

Header: `boost/asio/stream_socket_service.hpp`

Convenience header: `boost/asio.hpp`

stream_socket_service::get_io_service

Inherited from io_service.

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

stream_socket_service::get_option

Get a socket option.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename GettableSocketOption>
boost::system::error_code get_option(
    const implementation_type & impl,
    GettableSocketOption & option,
    boost::system::error_code & ec) const;
```

stream_socket_service::id

The unique service identifier.

```
static boost::asio::io_service::id id;
```

stream_socket_service::implementation_type

The type of a stream socket implementation.

```
typedef implementation_defined implementation_type;
```

Requirements

Header: `boost/asio/stream_socket_service.hpp`

Convenience header: `boost/asio.hpp`

stream_socket_service::io_control

Perform an IO control command on the socket.

```
template<
    typename IoControlCommand>
boost::system::error_code io_control(
    implementation_type & impl,
    IoControlCommand & command,
    boost::system::error_code & ec);
```

stream_socket_service::is_open

Determine whether the socket is open.

```
bool is_open(
    const implementation_type & impl) const;
```

stream_socket_service::local_endpoint

Get the local endpoint.

```
endpoint_type local_endpoint(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

stream_socket_service::move_assign

Move-assign from another stream socket implementation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void move_assign(
    implementation_type & impl,
    stream_socket_service & other_service,
    implementation_type & other_impl);
```

stream_socket_service::move_construct

Move-construct a new stream socket implementation.

```
void move_construct(
    implementation_type & impl,
    implementation_type & other_impl);
```

stream_socket_service::native

(Deprecated: Use `native_handle()`.) Get the native socket implementation.

```
native_type native(
    implementation_type & impl);
```

stream_socket_service::native_handle

Get the native socket implementation.

```
native_handle_type native_handle(
    implementation_type & impl);
```

stream_socket_service::native_handle_type

The native socket type.

```
typedef implementation_defined native_handle_type;
```

Requirements

Header: `boost/asio/stream_socket_service.hpp`

Convenience header: `boost/asio.hpp`

stream_socket_service::native_non_blocking

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking(
    const implementation_type & impl) const;
  » more...
```

Sets the non-blocking mode of the native socket implementation.

```
boost::system::error_code native_non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

stream_socket_service::native_non_blocking (1 of 2 overloads)

Gets the non-blocking mode of the native socket implementation.

```
bool native_non_blocking(
    const implementation_type & impl) const;
```

stream_socket_service::native_non_blocking (2 of 2 overloads)

Sets the non-blocking mode of the native socket implementation.

```
boost::system::error_code native_non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
```

stream_socket_service::native_type

(Deprecated: Use native_handle_type.) The native socket type.

```
typedef implementation_defined native_type;
```

Requirements

Header: `boost/asio/stream_socket_service.hpp`

Convenience header: `boost/asio.hpp`

stream_socket_service::non_blocking

Gets the non-blocking mode of the socket.

```
bool non_blocking(
    const implementation_type & impl) const;
  » more...
```

Sets the non-blocking mode of the socket.

```
boost::system::error_code non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
  » more...
```

stream_socket_service::non_blocking (1 of 2 overloads)

Gets the non-blocking mode of the socket.

```
bool non_blocking(
    const implementation_type & impl) const;
```

stream_socket_service::non_blocking (2 of 2 overloads)

Sets the non-blocking mode of the socket.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code non_blocking(
    implementation_type & impl,
    bool mode,
    boost::system::error_code & ec);
```

stream_socket_service::open

Open a stream socket.

```
boost::system::error_code open(
    implementation_type & impl,
    const protocol_type & protocol,
    boost::system::error_code & ec);
```

stream_socket_service::protocol_type

The protocol type.

```
typedef Protocol protocol_type;
```

Requirements

Header: `boost/asio/stream_socket_service.hpp`

Convenience header: `boost/asio.hpp`

stream_socket_service::receive

Receive some data from the peer.

```
template<
    typename MutableBufferSequence>
std::size_t receive(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

stream_socket_service::remote_endpoint

Get the remote endpoint.

```
endpoint_type remote_endpoint(
    const implementation_type & impl,
    boost::system::error_code & ec) const;
```

stream_socket_service::send

Send the given data to the peer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence>
std::size_t send(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    socket_base::message_flags flags,
    boost::system::error_code & ec);
```

stream_socket_service::set_option

Set a socket option.

```
template<
    typename SettableSocketOption>
boost::system::error_code set_option(
    implementation_type & impl,
    const SettableSocketOption & option,
    boost::system::error_code & ec);
```

stream_socket_service::shutdown

Disable sends or receives on the socket.

```
boost::system::error_code shutdown(
    implementation_type & impl,
    socket_base::shutdown_type what,
    boost::system::error_code & ec);
```

stream_socket_service::stream_socket_service

Construct a new stream socket service for the specified `io_service`.

```
stream_socket_service(
    boost::asio::io_service & io_service);
```

streambuf

Typedef for the typical usage of `basic_streambuf`.

```
typedef basic_streambuf streambuf;
```

Types

Name	Description
const_buffers_type	The type used to represent the input sequence as a list of buffers.
mutable_buffers_type	The type used to represent the output sequence as a list of buffers.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
basic_streambuf	Construct a basic_streambuf object.
commit	Move characters from the output sequence to the input sequence.
consume	Remove characters from the input sequence.
data	Get a list of buffers that represents the input sequence.
max_size	Get the maximum size of the basic_streambuf.
prepare	Get a list of buffers that represents the output sequence, with the given size.
size	Get the size of the input sequence.

Protected Member Functions

Name	Description
overflow	Override std::streambuf behaviour.
reserve	
underflow	Override std::streambuf behaviour.

The `basic_streambuf` class is derived from `std::streambuf` to associate the streambuf's input and output sequences with one or more character arrays. These character arrays are internal to the `basic_streambuf` object, but direct access to the array elements is provided to permit them to be used efficiently with I/O operations. Characters written to the output sequence of a `basic_streambuf` object are appended to the input sequence of the same object.

The `basic_streambuf` class's public interface is intended to permit the following implementation strategies:

- A single contiguous character array, which is reallocated as necessary to accommodate changes in the size of the character sequence. This is the implementation approach currently used in Asio.
- A sequence of one or more character arrays, where each array is of the same size. Additional character array objects are appended to the sequence to accommodate changes in the size of the character sequence.
- A sequence of one or more character arrays of varying sizes. Additional character array objects are appended to the sequence to accommodate changes in the size of the character sequence.

The constructor for `basic_streambuf` accepts a `size_t` argument specifying the maximum of the sum of the sizes of the input sequence and output sequence. During the lifetime of the `basic_streambuf` object, the following invariant holds:

```
size() <= max_size()
```

Any member function that would, if successful, cause the invariant to be violated shall throw an exception of class `std::length_error`.

The constructor for `basic_streambuf` takes an Allocator argument. A copy of this argument is used for any memory allocation performed, by the constructor and by all member functions, during the lifetime of each `basic_streambuf` object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Examples

Writing directly from an streambuf to a socket:

```
boost::asio::streambuf b;
std::ostream os(&b);
os << "Hello, World!\n";

// try sending some data in input sequence
size_t n = sock.send(b.data());

b.consume(n); // sent data is removed from input sequence
```

Reading from a socket directly into a streambuf:

```
boost::asio::streambuf b;

// reserve 512 bytes in output sequence
boost::asio::streambuf::mutable_buffers_type bufs = b.prepare(512);

size_t n = sock.receive(bufs);

// received data is "committed" from output sequence to input sequence
b.commit(n);

std::istream is(&b);
std::string s;
is >> s;
```

Requirements

Header: `boost/asio/streambuf.hpp`

Convenience header: `boost/asio.hpp`

system_timer

Typedef for a timer based on the system clock.

```
typedef basic_waitable_timer< chrono::system_clock > system_timer;
```

Types

Name	Description
clock_type	The clock type.
duration	The duration type of the clock.
implementation_type	The underlying implementation type of I/O object.
service_type	The type of the service that will be used to provide I/O operations.
time_point	The time point type of the clock.
traits_type	The wait traits type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
async_wait	Start an asynchronous wait on the timer.
basic_waitable_timer	Constructor. Constructor to set a particular expiry time as an absolute time. Constructor to set a particular expiry time relative to now.
cancel	Cancel any asynchronous operations that are waiting on the timer.
cancel_one	Cancels one asynchronous operation that is waiting on the timer.
expires_at	Get the timer's expiry time as an absolute time. Set the timer's expiry time as an absolute time.
expires_from_now	Get the timer's expiry time relative to now. Set the timer's expiry time relative to now.
get_io_service	Get the io_service associated with the object.
wait	Perform a blocking wait on the timer.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `basic_waitable_timer` class template provides the ability to perform a blocking or asynchronous wait for a timer to expire.

A waitable timer is always in one of two states: "expired" or "not expired". If the `wait()` or `async_wait()` function is called on an expired timer, the wait operation will complete immediately.

Most applications will use the boost::asio::waitable_timer typedef.

Remarks

This waitable timer functionality is for use with the C++11 standard library's `<chrono>` facility, or with the Boost.Chrono library.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Examples

Performing a blocking wait:

```
// Construct a timer without setting an expiry time.
boost::asio::waitable_timer timer(io_service);

// Set an expiry time relative to now.
timer.expires_from_now(boost::posix_time::seconds(5));

// Wait for the timer to expire.
timer.wait();
```

Performing an asynchronous wait:

```
void handler(const boost::system::error_code& error)
{
  if (!error)
  {
    // Timer expired.
  }
}

...

// Construct a timer with an absolute expiry time.
boost::asio::waitable_timer timer(io_service,
    boost::posix_time::time_from_string("2005-12-07 23:59:59.000"));

// Start an asynchronous wait.
timer.async_wait(handler);
```

Changing an active waitable_timer's expiry time

Changing the expiry time of a timer while there are pending asynchronous waits causes those wait operations to be cancelled. To ensure that the action associated with the timer is performed only once, use something like this: used:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void on_some_event()
{
  if (my_timer.expires_from_now(seconds(5)) > 0)
  {
    // We managed to cancel the timer. Start new asynchronous wait.
    my_timer.async_wait(on_timeout);
  }
  else
  {
    // Too late, timer has already expired!
  }
}

void on_timeout(const boost::system::error_code& e)
{
  if (e != boost::asio::error::operation_aborted)
  {
    // Timer was not cancelled, take necessary action.
  }
}
```

- The `boost::asio::basic_waitable_timer::expires_from_now()` function cancels any pending asynchronous waits, and returns the number of asynchronous waits that were cancelled. If it returns 0 then you were too late and the wait handler has already been executed, or will soon be executed. If it returns 1 then the wait handler was successfully cancelled.
- If a wait handler is cancelled, the boost::system::error_code passed to it contains the value `boost::asio::error::operation_aborted`.

This typedef uses the C++11 `<chrono>` standard library facility, if available. Otherwise, it may use the Boost.Chrono library. To explicitly utilise Boost.Chrono, use the `basic_waitable_timer` template directly:

```
typedef basic_waitable_timer<boost::chrono::system_clock> timer;
```

Requirements

Header: `boost/asio/system_timer.hpp`

Convenience header: None

time_traits< boost::posix_time::ptime >

Time traits specialised for posix_time.

```
template<>
struct time_traits< boost::posix_time::ptime >
```

Types

Name	Description
duration_type	The duration type.
time_type	The time type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
add	Add a duration to a time.
less_than	Test whether one time is less than another.
now	Get the current time.
subtract	Subtract one time from another.
to_posix_duration	Convert to POSIX duration type.

Requirements

Header: `boost/asio/time_traits.hpp`

Convenience header: `boost/asio.hpp`

time_traits< boost::posix_time::ptime >::add

Add a duration to a time.

```
static time_type add(
    const time_type & t,
    const duration_type & d);
```

time_traits< boost::posix_time::ptime >::duration_type

The duration type.

```
typedef boost::posix_time::time_duration duration_type;
```

Requirements

Header: `boost/asio/time_traits.hpp`

Convenience header: `boost/asio.hpp`

time_traits< boost::posix_time::ptime >::less_than

Test whether one time is less than another.

```
static bool less_than(
    const time_type & t1,
    const time_type & t2);
```

time_traits< boost::posix_time::ptime >::now

Get the current time.

```
static time_type now();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

time_traits< boost::posix_time::ptime >::subtract

Subtract one time from another.

```
static duration_type subtract(
    const time_type & t1,
    const time_type & t2);
```

time_traits< boost::posix_time::ptime >::time_type

The time type.

```
typedef boost::posix_time::ptime time_type;
```

Requirements

Header: `boost/asio/time_traits.hpp`

Convenience header: `boost/asio.hpp`

time_traits< boost::posix_time::ptime >::to_posix_duration

Convert to POSIX duration type.

```
static boost::posix_time::time_duration to_posix_duration(
    const duration_type & d);
```

transfer_all

Return a completion condition function object that indicates that a read or write operation should continue until all of the data has been transferred, or until an error occurs.

```
unspecified transfer_all();
```

This function is used to create an object, of unspecified type, that meets CompletionCondition requirements.

Example

Reading until a buffer is full:

```
boost::array<char, 128> buf;
boost::system::error_code ec;
std::size_t n = boost::asio::read(
    sock, boost::asio::buffer(buf),
    boost::asio::transfer_all(), ec);
if (ec)
{
  // An error occurred.
}
else
{
  // n == 128
}
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/completion_condition.hpp`

Convenience header: `boost/asio.hpp`

transfer_at_least

Return a completion condition function object that indicates that a read or write operation should continue until a minimum number of bytes has been transferred, or until an error occurs.

```
unspecified transfer_at_least(
    std::size_t minimum);
```

This function is used to create an object, of unspecified type, that meets CompletionCondition requirements.

Example

Reading until a buffer is full or contains at least 64 bytes:

```
boost::array<char, 128> buf;
boost::system::error_code ec;
std::size_t n = boost::asio::read(
    sock, boost::asio::buffer(buf),
    boost::asio::transfer_at_least(64), ec);
if (ec)
{
  // An error occurred.
}
else
{
  // n >= 64 && n <= 128
}
```

Requirements

Header: `boost/asio/completion_condition.hpp`

Convenience header: `boost/asio.hpp`

transfer_exactly

Return a completion condition function object that indicates that a read or write operation should continue until an exact number of bytes has been transferred, or until an error occurs.

```
unspecified transfer_exactly(
    std::size_t size);
```

This function is used to create an object, of unspecified type, that meets CompletionCondition requirements.

Example

Reading until a buffer is full or contains exactly 64 bytes:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::array<char, 128> buf;
boost::system::error_code ec;
std::size_t n = boost::asio::read(
    sock, boost::asio::buffer(buf),
    boost::asio::transfer_exactly(64), ec);
if (ec)
{
  // An error occurred.
}
else
{
  // n == 64
}
```

Requirements

Header: `boost/asio/completion_condition.hpp`

Convenience header: `boost/asio.hpp`

use_service

```
template<
    typename Service>
Service & use_service(
    io_service & ios);
```

This function is used to locate a service object that corresponds to the given service type. If there is no existing implementation of the service, then the `io_service` will create a new instance of the service.

Parameters

ios The `io_service` object that owns the service.

Return Value

The service interface implementing the specified service type. Ownership of the service interface is not transferred to the caller.

Requirements

Header: `boost/asio/io_service.hpp`

Convenience header: `boost/asio.hpp`

wait_traits

Wait traits suitable for use with the `basic_waitable_timer` class template.

```
template<
    typename Clock>
struct wait_traits
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
to_wait_duration	Convert a clock duration into a duration used for waiting.

Requirements

Header: `boost/asio/wait_traits.hpp`

Convenience header: `boost/asio.hpp`

wait_traits::to_wait_duration

Convert a clock duration into a duration used for waiting.

```
static Clock::duration to_wait_duration(
    const typename Clock::duration & d);
```

Return Value

d.

waitable_timer_service

Default service implementation for a timer.

```
template<
    typename Clock,
    typename WaitTraits = boost::asio::wait_traits<Clock>>
class waitable_timer_service :
  public io_service::service
```

Types

Name	Description
clock_type	The clock type.
duration	The duration type of the clock.
implementation_type	The implementation type of the waitable timer.
time_point	The time point type of the clock.
traits_type	The wait traits type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
async_wait	
cancel	Cancel any asynchronous wait operations associated with the timer.
cancel_one	Cancels one asynchronous wait operation associated with the timer.
construct	Construct a new timer implementation.
destroy	Destroy a timer implementation.
expires_at	Get the expiry time for the timer as an absolute time. Set the expiry time for the timer as an absolute time.
expires_from_now	Get the expiry time for the timer relative to now. Set the expiry time for the timer relative to now.
get_io_service	Get the io_service object that owns the service.
wait	
waitable_timer_service	Construct a new timer service for the specified io_service.

Data Members

Name	Description
id	The unique service identifier.

Requirements

Header: `boost/asio/waitable_timer_service.hpp`

Convenience header: `boost/asio.hpp`

waitable_timer_service::async_wait

```
template<
    typename WaitHandler>
void async_wait(
    implementation_type & impl,
    WaitHandler handler);
```

waitable_timer_service::cancel

Cancel any asynchronous wait operations associated with the timer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t cancel(
    implementation_type & impl,
    boost::system::error_code & ec);
```

waitable_timer_service::cancel_one

Cancels one asynchronous wait operation associated with the timer.

```
std::size_t cancel_one(
    implementation_type & impl,
    boost::system::error_code & ec);
```

waitable_timer_service::clock_type

The clock type.

```
typedef Clock clock_type;
```

Requirements

Header: `boost/asio/waitable_timer_service.hpp`

Convenience header: `boost/asio.hpp`

waitable_timer_service::construct

Construct a new timer implementation.

```
void construct(
    implementation_type & impl);
```

waitable_timer_service::destroy

Destroy a timer implementation.

```
void destroy(
    implementation_type & impl);
```

waitable_timer_service::duration

The duration type of the clock.

```
typedef clock_type::duration duration;
```

Requirements

Header: `boost/asio/waitable_timer_service.hpp`

Convenience header: `boost/asio.hpp`

waitable_timer_service::expires_at

Get the expiry time for the timer as an absolute time.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
time_point expires_at(
    const implementation_type & impl) const;
  » more...
```

Set the expiry time for the timer as an absolute time.

```
std::size_t expires_at(
    implementation_type & impl,
    const time_point & expiry_time,
    boost::system::error_code & ec);
  » more...
```

waitable_timer_service::expires_at (1 of 2 overloads)

Get the expiry time for the timer as an absolute time.

```
time_point expires_at(
    const implementation_type & impl) const;
```

waitable_timer_service::expires_at (2 of 2 overloads)

Set the expiry time for the timer as an absolute time.

```
std::size_t expires_at(
    implementation_type & impl,
    const time_point & expiry_time,
    boost::system::error_code & ec);
```

waitable_timer_service::expires_from_now

Get the expiry time for the timer relative to now.

```
duration expires_from_now(
    const implementation_type & impl) const;
  » more...
```

Set the expiry time for the timer relative to now.

```
std::size_t expires_from_now(
    implementation_type & impl,
    const duration & expiry_time,
    boost::system::error_code & ec);
  » more...
```

waitable_timer_service::expires_from_now (1 of 2 overloads)

Get the expiry time for the timer relative to now.

```
duration expires_from_now(
    const implementation_type & impl) const;
```

waitable_timer_service::expires_from_now (2 of 2 overloads)

Set the expiry time for the timer relative to now.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t expires_from_now(
    implementation_type & impl,
    const duration & expiry_time,
    boost::system::error_code & ec);
```

waitable_timer_service::get_io_service

Inherited from io_service.

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

waitable_timer_service::id

The unique service identifier.

```
static boost::asio::io_service::id id;
```

waitable_timer_service::implementation_type

The implementation type of the waitable timer.

```
typedef implementation_defined implementation_type;
```

Requirements

Header: `boost/asio/waitable_timer_service.hpp`

Convenience header: `boost/asio.hpp`

waitable_timer_service::time_point

The time point type of the clock.

```
typedef clock_type::time_point time_point;
```

Requirements

Header: `boost/asio/waitable_timer_service.hpp`

Convenience header: `boost/asio.hpp`

waitable_timer_service::traits_type

The wait traits type.

```
typedef WaitTraits traits_type;
```

Requirements

Header: `boost/asio/waitable_timer_service.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

waitable_timer_service::wait

```
void wait(
    implementation_type & impl,
    boost::system::error_code & ec);
```

waitable_timer_service::waitable_timer_service

Construct a new timer service for the specified `io_service`.

```
waitable_timer_service(
    boost::asio::io_service & io_service);
```

windows::basic_handle

Provides Windows handle functionality.

```
template<
    typename HandleService>
class basic_handle :
  public basic_io_object< HandleService >
```

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_handle is always the lowest layer.
native_handle_type	The native representation of a handle.
native_type	(Deprecated: Use native_handle_type.) The native representation of a handle.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native handle to the handle.
basic_handle	Construct a basic_handle without opening it. Construct a basic_handle on an existing native handle. Move-construct a basic_handle from another.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close the handle.
get_io_service	Get the io_service associated with the object.
is_open	Determine whether the handle is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native handle representation.
native_handle	Get the native handle representation.
operator=	Move-assign a basic_handle from another.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_handle	Protected destructor to prevent deletion through this type.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `windows::basic_handle` class template provides the ability to wrap a Windows handle.

Thread Safety

Distinct objects: Safe.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Shared objects: Unsafe.

Requirements

Header: `boost/asio/windows/basic_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_handle::assign

Assign an existing native handle to the handle.

```
void assign(
    const native_handle_type & handle);
  » more...

boost::system::error_code assign(
    const native_handle_type & handle,
    boost::system::error_code & ec);
  » more...
```

windows::basic_handle::assign (1 of 2 overloads)

Assign an existing native handle to the handle.

```
void assign(
    const native_handle_type & handle);
```

windows::basic_handle::assign (2 of 2 overloads)

Assign an existing native handle to the handle.

```
boost::system::error_code assign(
    const native_handle_type & handle,
    boost::system::error_code & ec);
```

windows::basic_handle::basic_handle

Construct a `windows::basic_handle` without opening it.

```
explicit basic_handle(
    boost::asio::io_service & io_service);
  » more...
```

Construct a `windows::basic_handle` on an existing native handle.

```
basic_handle(
    boost::asio::io_service & io_service,
    const native_handle_type & handle);
  » more...
```

Move-construct a `windows::basic_handle` from another.

```
basic_handle(
    basic_handle && other);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_handle::basic_handle (1 of 3 overloads)

Construct a `windows::basic_handle` without opening it.

```
basic_handle(
    boost::asio::io_service & io_service);
```

This constructor creates a handle without opening it.

Parameters

io_service — The `io_service` object that the handle will use to dispatch handlers for any asynchronous operations performed on the handle.

windows::basic_handle::basic_handle (2 of 3 overloads)

Construct a `windows::basic_handle` on an existing native handle.

```
basic_handle(
    boost::asio::io_service & io_service,
    const native_handle_type & handle);
```

This constructor creates a handle object to hold an existing native handle.

Parameters

io_service — The `io_service` object that the handle will use to dispatch handlers for any asynchronous operations performed on the handle.

handle — A native handle.

Exceptions

boost::system::system_error — Thrown on failure.

windows::basic_handle::basic_handle (3 of 3 overloads)

Move-construct a `windows::basic_handle` from another.

```
basic_handle(
    basic_handle && other);
```

This constructor moves a handle from one object to another.

Parameters

other — The other `windows::basic_handle` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_handle(io_service&) constructor`.

windows::basic_handle::cancel

Cancel all asynchronous operations associated with the handle.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

windows::basic_handle::cancel (1 of 2 overloads)

Cancel all asynchronous operations associated with the handle.

```
void cancel();
```

This function causes all outstanding asynchronous read or write operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error Thrown on failure.

windows::basic_handle::cancel (2 of 2 overloads)

Cancel all asynchronous operations associated with the handle.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function causes all outstanding asynchronous read or write operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Parameters

ec Set to indicate what error occurred, if any.

windows::basic_handle::close

Close the handle.

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

windows::basic_handle::close (1 of 2 overloads)

Close the handle.

```
void close();
```

This function is used to close the handle. Any asynchronous read or write operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

windows::basic_handle::close (2 of 2 overloads)

Close the handle.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```

This function is used to close the handle. Any asynchronous read or write operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Parameters

ec	Set to indicate what error occurred, if any.

windows::basic_handle::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

windows::basic_handle::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

windows::basic_handle::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

windows::basic_handle::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

windows::basic_handle::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

windows::basic_handle::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

windows::basic_handle::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

windows::basic_handle::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

windows::basic_handle::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/windows/basic_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_handle::is_open

Determine whether the handle is open.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
bool is_open() const;
```

windows::basic_handle::lowest_layer

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

windows::basic_handle::lowest_layer (1 of 2 overloads)

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

This function returns a reference to the lowest layer in a stack of layers. Since a `windows::basic_handle` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

windows::basic_handle::lowest_layer (2 of 2 overloads)

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

This function returns a const reference to the lowest layer in a stack of layers. Since a `windows::basic_handle` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A const reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

windows::basic_handle::lowest_layer_type

A `windows::basic_handle` is always the lowest layer.

```
typedef basic_handle< HandleService > lowest_layer_type;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_handle is always the lowest layer.
native_handle_type	The native representation of a handle.
native_type	(Deprecated: Use native_handle_type.) The native representation of a handle.
service_type	The type of the service that will be used to provide I/O operations.

Member Functions

Name	Description
assign	Assign an existing native handle to the handle.
basic_handle	Construct a basic_handle without opening it. Construct a basic_handle on an existing native handle. Move-construct a basic_handle from another.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close the handle.
get_io_service	Get the io_service associated with the object.
is_open	Determine whether the handle is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native handle representation.
native_handle	Get the native handle representation.
operator=	Move-assign a basic_handle from another.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_handle	Protected destructor to prevent deletion through this type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `windows::basic_handle` class template provides the ability to wrap a Windows handle.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/windows/basic_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_handle::native

(Deprecated: Use `native_handle()`.) Get the native handle representation.

```
native_type native();
```

This function may be used to obtain the underlying representation of the handle. This is intended to allow access to native handle functionality that is not otherwise provided.

windows::basic_handle::native_handle

Get the native handle representation.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying representation of the handle. This is intended to allow access to native handle functionality that is not otherwise provided.

windows::basic_handle::native_handle_type

The native representation of a handle.

```
typedef HandleService::native_handle_type native_handle_type;
```

Requirements

Header: `boost/asio/windows/basic_handle.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_handle::native_type

(Deprecated: Use native_handle_type.) The native representation of a handle.

```
typedef HandleService::native_handle_type native_type;
```

Requirements

Header: `boost/asio/windows/basic_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_handle::operator=

Move-assign a `windows::basic_handle` from another.

```
basic_handle & operator=(
    basic_handle && other);
```

This assignment operator moves a handle from one object to another.

Parameters

other The other `windows::basic_handle` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_handle(io_service&)` constructor.

windows::basic_handle::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

windows::basic_handle::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef HandleService service_type;
```

Requirements

Header: `boost/asio/windows/basic_handle.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_handle::~basic_handle

Protected destructor to prevent deletion through this type.

```
~basic_handle();
```

windows::basic_object_handle

Provides object-oriented handle functionality.

```
template<
    typename ObjectHandleService = object_handle_service>
class basic_object_handle :
  public windows::basic_handle< ObjectHandleService >
```

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_handle is always the lowest layer.
native_handle_type	The native representation of a handle.
native_type	(Deprecated: Use native_handle_type.) The native representation of a handle.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native handle to the handle.
async_wait	Start an asynchronous wait on the object handle.
basic_object_handle	Construct a basic_object_handle without opening it. Construct a basic_object_handle on an existing native handle. Move-construct a basic_object_handle from another.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close the handle.
get_io_service	Get the io_service associated with the object.
is_open	Determine whether the handle is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native handle representation.
native_handle	Get the native handle representation.
operator=	Move-assign a basic_object_handle from another.
wait	Perform a blocking wait on the object handle.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `windows::basic_object_handle` class template provides asynchronous and blocking object-oriented handle functionality.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/windows/basic_object_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_object_handle::assign

Assign an existing native handle to the handle.

```
void assign(
    const native_handle_type & handle);
  » more...

boost::system::error_code assign(
    const native_handle_type & handle,
    boost::system::error_code & ec);
  » more...
```

windows::basic_object_handle::assign (1 of 2 overloads)

Inherited from windows::basic_handle.

Assign an existing native handle to the handle.

```
void assign(
    const native_handle_type & handle);
```

windows::basic_object_handle::assign (2 of 2 overloads)

Inherited from windows::basic_handle.

Assign an existing native handle to the handle.

```
boost::system::error_code assign(
    const native_handle_type & handle,
    boost::system::error_code & ec);
```

windows::basic_object_handle::async_wait

Start an asynchronous wait on the object handle.

```
template<
    typename WaitHandler>
void async_wait(
    WaitHandler handler);
```

This function is be used to initiate an asynchronous wait against the object handle. It always returns immediately.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

handler — The handler to be called when the object handle is set to the signalled state. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error // Result of operation.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

windows::basic_object_handle::basic_object_handle

Construct a `windows::basic_object_handle` without opening it.

```
explicit basic_object_handle(
    boost::asio::io_service & io_service);
  » more...
```

Construct a `windows::basic_object_handle` on an existing native handle.

```
basic_object_handle(
    boost::asio::io_service & io_service,
    const native_handle_type & native_handle);
  » more...
```

Move-construct a `windows::basic_object_handle` from another.

```
basic_object_handle(
    basic_object_handle && other);
  » more...
```

windows::basic_object_handle::basic_object_handle (1 of 3 overloads)

Construct a `windows::basic_object_handle` without opening it.

```
basic_object_handle(
    boost::asio::io_service & io_service);
```

This constructor creates an object handle without opening it.

Parameters

io_service — The `io_service` object that the object handle will use to dispatch handlers for any asynchronous operations performed on the handle.

windows::basic_object_handle::basic_object_handle (2 of 3 overloads)

Construct a `windows::basic_object_handle` on an existing native handle.

```
basic_object_handle(
    boost::asio::io_service & io_service,
    const native_handle_type & native_handle);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This constructor creates an object handle object to hold an existing native handle.

Parameters

io_service	The `io_service` object that the object handle will use to dispatch handlers for any asynchronous operations performed on the handle.
native_handle	The new underlying handle implementation.

Exceptions

boost::system::system_error	Thrown on failure.

windows::basic_object_handle::basic_object_handle (3 of 3 overloads)

Move-construct a `windows::basic_object_handle` from another.

```
basic_object_handle(
    basic_object_handle && other);
```

This constructor moves an object handle from one object to another.

Parameters

other	The other `windows::basic_object_handle` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_object_handle(io_service&) constructor`.

windows::basic_object_handle::cancel

Cancel all asynchronous operations associated with the handle.

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

windows::basic_object_handle::cancel (1 of 2 overloads)

Inherited from windows::basic_handle.

Cancel all asynchronous operations associated with the handle.

```
void cancel();
```

This function causes all outstanding asynchronous read or write operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_object_handle::cancel (2 of 2 overloads)

Inherited from windows::basic_handle.

Cancel all asynchronous operations associated with the handle.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function causes all outstanding asynchronous read or write operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Parameters

ec Set to indicate what error occurred, if any.

windows::basic_object_handle::close

Close the handle.

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

windows::basic_object_handle::close (1 of 2 overloads)

Inherited from windows::basic_handle.

Close the handle.

```
void close();
```

This function is used to close the handle. Any asynchronous read or write operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error Thrown on failure.

windows::basic_object_handle::close (2 of 2 overloads)

Inherited from windows::basic_handle.

Close the handle.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```

This function is used to close the handle. Any asynchronous read or write operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Parameters

ec Set to indicate what error occurred, if any.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_object_handle::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

windows::basic_object_handle::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```

windows::basic_object_handle::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

windows::basic_object_handle::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

windows::basic_object_handle::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

windows::basic_object_handle::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
service_type & get_service();
```

windows::basic_object_handle::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

windows::basic_object_handle::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```

windows::basic_object_handle::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/windows/basic_object_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_object_handle::is_open

Inherited from windows::basic_handle.

Determine whether the handle is open.

```
bool is_open() const;
```

windows::basic_object_handle::lowest_layer

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_object_handle::lowest_layer (1 of 2 overloads)

Inherited from windows::basic_handle.

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

This function returns a reference to the lowest layer in a stack of layers. Since a `windows::basic_handle` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

windows::basic_object_handle::lowest_layer (2 of 2 overloads)

Inherited from windows::basic_handle.

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
```

This function returns a const reference to the lowest layer in a stack of layers. Since a `windows::basic_handle` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A const reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

windows::basic_object_handle::lowest_layer_type

Inherited from windows::basic_handle.

A `windows::basic_handle` is always the lowest layer.

```
typedef basic_handle< ObjectHandleService > lowest_layer_type;
```

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_handle is always the lowest layer.
native_handle_type	The native representation of a handle.
native_type	(Deprecated: Use native_handle_type.) The native representation of a handle.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native handle to the handle.
basic_handle	Construct a basic_handle without opening it. Construct a basic_handle on an existing native handle. Move-construct a basic_handle from another.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close the handle.
get_io_service	Get the io_service associated with the object.
is_open	Determine whether the handle is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native handle representation.
native_handle	Get the native handle representation.
operator=	Move-assign a basic_handle from another.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_handle	Protected destructor to prevent deletion through this type.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `windows::basic_handle` class template provides the ability to wrap a Windows handle.

Thread Safety

Distinct objects: Safe.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Shared objects: Unsafe.

Requirements

Header: `boost/asio/windows/basic_object_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_object_handle::native

Inherited from windows::basic_handle.

(Deprecated: Use `native_handle()`.) Get the native handle representation.

```
native_type native();
```

This function may be used to obtain the underlying representation of the handle. This is intended to allow access to native handle functionality that is not otherwise provided.

windows::basic_object_handle::native_handle

Inherited from windows::basic_handle.

Get the native handle representation.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying representation of the handle. This is intended to allow access to native handle functionality that is not otherwise provided.

windows::basic_object_handle::native_handle_type

The native representation of a handle.

```
typedef ObjectHandleService::native_handle_type native_handle_type;
```

Requirements

Header: `boost/asio/windows/basic_object_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_object_handle::native_type

Inherited from windows::basic_handle.

(Deprecated: Use native_handle_type.) The native representation of a handle.

```
typedef ObjectHandleService::native_handle_type native_type;
```

Requirements

Header: `boost/asio/windows/basic_object_handle.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_object_handle::operator=

Move-assign a `windows::basic_object_handle` from another.

```
basic_object_handle & operator=(
    basic_object_handle && other);
```

This assignment operator moves an object handle from one object to another.

Parameters

other The other `windows::basic_object_handle` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_object_handle(io_service&)` `constructor`.

windows::basic_object_handle::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

windows::basic_object_handle::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef ObjectHandleService service_type;
```

Requirements

Header: `boost/asio/windows/basic_object_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_object_handle::wait

Perform a blocking wait on the object handle.

```
void wait();
  » more...

void wait(
    boost::system::error_code & ec);
  » more...
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_object_handle::wait (1 of 2 overloads)

Perform a blocking wait on the object handle.

```
void wait();
```

This function is used to wait for the object handle to be set to the signalled state. This function blocks and does not return until the object handle has been set to the signalled state.

Exceptions

boost::system::system_error Thrown on failure.

windows::basic_object_handle::wait (2 of 2 overloads)

Perform a blocking wait on the object handle.

```
void wait(
    boost::system::error_code & ec);
```

This function is used to wait for the object handle to be set to the signalled state. This function blocks and does not return until the object handle has been set to the signalled state.

Parameters

ec Set to indicate what error occurred, if any.

windows::basic_random_access_handle

Provides random-access handle functionality.

```
template<
    typename RandomAccessHandleService = random_access_handle_service>
class basic_random_access_handle :
  public windows::basic_handle< RandomAccessHandleService >
```

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_handle is always the lowest layer.
native_handle_type	The native representation of a handle.
native_type	(Deprecated: Use native_handle_type.) The native representation of a handle.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native handle to the handle.
async_read_some_at	Start an asynchronous read at the specified offset.
async_write_some_at	Start an asynchronous write at the specified offset.
basic_random_access_handle	Construct a basic_random_access_handle without opening it. Construct a basic_random_access_handle on an existing native handle. Move-construct a basic_random_access_handle from another.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close the handle.
get_io_service	Get the io_service associated with the object.
is_open	Determine whether the handle is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native handle representation.
native_handle	Get the native handle representation.
operator=	Move-assign a basic_random_access_handle from another.
read_some_at	Read some data from the handle at the specified offset.
write_some_at	Write some data to the handle at the specified offset.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `windows::basic_random_access_handle` class template provides asynchronous and blocking random-access handle functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/windows/basic_random_access_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_random_access_handle::assign

Assign an existing native handle to the handle.

```
void assign(
    const native_handle_type & handle);
  » more...

boost::system::error_code assign(
    const native_handle_type & handle,
    boost::system::error_code & ec);
  » more...
```

windows::basic_random_access_handle::assign (1 of 2 overloads)

Inherited from windows::basic_handle.

Assign an existing native handle to the handle.

```
void assign(
    const native_handle_type & handle);
```

windows::basic_random_access_handle::assign (2 of 2 overloads)

Inherited from windows::basic_handle.

Assign an existing native handle to the handle.

```
boost::system::error_code assign(
    const native_handle_type & handle,
    boost::system::error_code & ec);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_random_access_handle::async_read_some_at

Start an asynchronous read at the specified offset.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_some_at(
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

This function is used to asynchronously read data from the random-access handle. The function call always returns immediately.

Parameters

offset — The offset at which the data will be read.

buffers — One or more buffers into which the data will be read. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes read.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The read operation may not read all of the requested number of bytes. Consider using the `async_read_at` function if you need to ensure that the requested amount of data is read before the asynchronous operation completes.

Example

To read into a single data buffer use the `buffer` function as follows:

```
handle.async_read_some_at(42, boost::asio::buffer(data, size), handler);
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

windows::basic_random_access_handle::async_write_some_at

Start an asynchronous write at the specified offset.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_some_at(
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

This function is used to asynchronously write data to the random-access handle. The function call always returns immediately.

Parameters

offset — The offset at which the data will be written.

buffers — One or more data buffers to be written to the handle. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the write operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes written.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The write operation may not transmit all of the data to the peer. Consider using the `async_write_at` function if you need to ensure that all data is written before the asynchronous operation completes.

Example

To write a single data buffer use the `buffer` function as follows:

```
handle.async_write_some_at(42, boost::asio::buffer(data, size), handler);
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

windows::basic_random_access_handle::basic_random_access_handle

Construct a `windows::basic_random_access_handle` without opening it.

```
explicit basic_random_access_handle(
    boost::asio::io_service & io_service);
  » more...
```

Construct a `windows::basic_random_access_handle` on an existing native handle.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
basic_random_access_handle(
    boost::asio::io_service & io_service,
    const native_handle_type & handle);
  » more...
```

Move-construct a `windows::basic_random_access_handle` from another.

```
basic_random_access_handle(
    basic_random_access_handle && other);
  » more...
```

windows::basic_random_access_handle::basic_random_access_handle (1 of 3 overloads)

Construct a `windows::basic_random_access_handle` without opening it.

```
basic_random_access_handle(
    boost::asio::io_service & io_service);
```

This constructor creates a random-access handle without opening it. The handle needs to be opened before data can be written to or or read from it.

Parameters

io_service — The `io_service` object that the random-access handle will use to dispatch handlers for any asynchronous operations performed on the handle.

windows::basic_random_access_handle::basic_random_access_handle (2 of 3 overloads)

Construct a `windows::basic_random_access_handle` on an existing native handle.

```
basic_random_access_handle(
    boost::asio::io_service & io_service,
    const native_handle_type & handle);
```

This constructor creates a random-access handle object to hold an existing native handle.

Parameters

io_service — The `io_service` object that the random-access handle will use to dispatch handlers for any asynchronous operations performed on the handle.

handle — The new underlying handle implementation.

Exceptions

boost::system::system_error — Thrown on failure.

windows::basic_random_access_handle::basic_random_access_handle (3 of 3 overloads)

Move-construct a `windows::basic_random_access_handle` from another.

```
basic_random_access_handle(
    basic_random_access_handle && other);
```

This constructor moves a random-access handle from one object to another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

other The other `windows::basic_random_access_handle` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_random_access_handle(io_service&)` constructor.

windows::basic_random_access_handle::cancel

Cancel all asynchronous operations associated with the handle.

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

windows::basic_random_access_handle::cancel (1 of 2 overloads)

Inherited from windows::basic_handle.

Cancel all asynchronous operations associated with the handle.

```
void cancel();
```

This function causes all outstanding asynchronous read or write operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error Thrown on failure.

windows::basic_random_access_handle::cancel (2 of 2 overloads)

Inherited from windows::basic_handle.

Cancel all asynchronous operations associated with the handle.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function causes all outstanding asynchronous read or write operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Parameters

ec Set to indicate what error occurred, if any.

windows::basic_random_access_handle::close

Close the handle.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

windows::basic_random_access_handle::close (1 of 2 overloads)

Inherited from windows::basic_handle.

Close the handle.

```
void close();
```

This function is used to close the handle. Any asynchronous read or write operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure.

windows::basic_random_access_handle::close (2 of 2 overloads)

Inherited from windows::basic_handle.

Close the handle.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```

This function is used to close the handle. Any asynchronous read or write operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Parameters

ec	Set to indicate what error occurred, if any.

windows::basic_random_access_handle::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

windows::basic_random_access_handle::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_random_access_handle::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

windows::basic_random_access_handle::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

windows::basic_random_access_handle::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

windows::basic_random_access_handle::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

windows::basic_random_access_handle::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

windows::basic_random_access_handle::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_random_access_handle::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/windows/basic_random_access_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_random_access_handle::is_open

Inherited from windows::basic_handle.

Determine whether the handle is open.

```
bool is_open() const;
```

windows::basic_random_access_handle::lowest_layer

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

windows::basic_random_access_handle::lowest_layer (1 of 2 overloads)

Inherited from windows::basic_handle.

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

This function returns a reference to the lowest layer in a stack of layers. Since a `windows::basic_handle` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

windows::basic_random_access_handle::lowest_layer (2 of 2 overloads)

Inherited from windows::basic_handle.

Get a const reference to the lowest layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
const lowest_layer_type & lowest_layer() const;
```

This function returns a const reference to the lowest layer in a stack of layers. Since a `windows::basic_handle` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A const reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

windows::basic_random_access_handle::lowest_layer_type

Inherited from windows::basic_handle.

A `windows::basic_handle` is always the lowest layer.

```
typedef basic_handle< RandomAccessHandleService > lowest_layer_type;
```

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_handle is always the lowest layer.
native_handle_type	The native representation of a handle.
native_type	(Deprecated: Use native_handle_type.) The native representation of a handle.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native handle to the handle.
basic_handle	Construct a basic_handle without opening it. Construct a basic_handle on an existing native handle. Move-construct a basic_handle from another.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close the handle.
get_io_service	Get the io_service associated with the object.
is_open	Determine whether the handle is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native handle representation.
native_handle	Get the native handle representation.
operator=	Move-assign a basic_handle from another.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_handle	Protected destructor to prevent deletion through this type.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `windows::basic_handle` class template provides the ability to wrap a Windows handle.

Thread Safety

Distinct objects: Safe.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Shared objects: Unsafe.

Requirements

Header: `boost/asio/windows/basic_random_access_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_random_access_handle::native

Inherited from windows::basic_handle.

(Deprecated: Use `native_handle()`.) Get the native handle representation.

```
native_type native();
```

This function may be used to obtain the underlying representation of the handle. This is intended to allow access to native handle functionality that is not otherwise provided.

windows::basic_random_access_handle::native_handle

Inherited from windows::basic_handle.

Get the native handle representation.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying representation of the handle. This is intended to allow access to native handle functionality that is not otherwise provided.

windows::basic_random_access_handle::native_handle_type

The native representation of a handle.

```
typedef RandomAccessHandleService::native_handle_type native_handle_type;
```

Requirements

Header: `boost/asio/windows/basic_random_access_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_random_access_handle::native_type

(Deprecated: Use native_handle_type.) The native representation of a handle.

```
typedef RandomAccessHandleService::native_handle_type native_type;
```

Requirements

Header: `boost/asio/windows/basic_random_access_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_random_access_handle::operator=

Move-assign a `windows::basic_random_access_handle` from another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
basic_random_access_handle & operator=(
    basic_random_access_handle && other);
```

This assignment operator moves a random-access handle from one object to another.

Parameters

other The other `windows::basic_random_access_handle` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_random_access_handle(io_service&)` constructor.

windows::basic_random_access_handle::read_some_at

Read some data from the handle at the specified offset.

```
template<
    typename MutableBufferSequence>
std::size_t read_some_at(
    boost::uint64_t offset,
    const MutableBufferSequence & buffers);
  » more...

template<
    typename MutableBufferSequence>
std::size_t read_some_at(
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

windows::basic_random_access_handle::read_some_at (1 of 2 overloads)

Read some data from the handle at the specified offset.

```
template<
    typename MutableBufferSequence>
std::size_t read_some_at(
    boost::uint64_t offset,
    const MutableBufferSequence & buffers);
```

This function is used to read data from the random-access handle. The function call will block until one or more bytes of data has been read successfully, or until an error occurs.

Parameters

offset The offset at which the data will be read.

buffers One or more buffers into which the data will be read.

Return Value

The number of bytes read.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure. An error code of `boost::asio::error::eof` indicates that the connection was closed by the peer.

Remarks

The read_some operation may not read all of the requested number of bytes. Consider using the `read_at` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

Example

To read into a single data buffer use the `buffer` function as follows:

```
handle.read_some_at(42, boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

windows::basic_random_access_handle::read_some_at (2 of 2 overloads)

Read some data from the handle at the specified offset.

```
template<
    typename MutableBufferSequence>
std::size_t read_some_at(
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to read data from the random-access handle. The function call will block until one or more bytes of data has been read successfully, or until an error occurs.

Parameters

offset	The offset at which the data will be read.
buffers	One or more buffers into which the data will be read.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes read. Returns 0 if an error occurred.

Remarks

The read_some operation may not read all of the requested number of bytes. Consider using the `read_at` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

windows::basic_random_access_handle::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

Available only for services that do not support movability.

windows::basic_random_access_handle::service_type

Inherited from basic_io_object.

The type of the service that will be used to provide I/O operations.

```
typedef RandomAccessHandleService service_type;
```

Requirements

Header: `boost/asio/windows/basic_random_access_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_random_access_handle::write_some_at

Write some data to the handle at the specified offset.

```
template<
    typename ConstBufferSequence>
std::size_t write_some_at(
    boost::uint64_t offset,
    const ConstBufferSequence & buffers);
  » more...

template<
    typename ConstBufferSequence>
std::size_t write_some_at(
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

windows::basic_random_access_handle::write_some_at (1 of 2 overloads)

Write some data to the handle at the specified offset.

```
template<
    typename ConstBufferSequence>
std::size_t write_some_at(
    boost::uint64_t offset,
    const ConstBufferSequence & buffers);
```

This function is used to write data to the random-access handle. The function call will block until one or more bytes of the data has been written successfully, or until an error occurs.

Parameters

offset	The offset at which the data will be written.
buffers	One or more data buffers to be written to the handle.

Return Value

The number of bytes written.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure. An error code of `boost::asio::error::eof` indicates that the connection was closed by the peer.

Remarks

The write_some_at operation may not write all of the data. Consider using the `write_at` function if you need to ensure that all data is written before the blocking operation completes.

Example

To write a single data buffer use the `buffer` function as follows:

```
handle.write_some_at(42, boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

windows::basic_random_access_handle::write_some_at (2 of 2 overloads)

Write some data to the handle at the specified offset.

```
template<
    typename ConstBufferSequence>
std::size_t write_some_at(
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to write data to the random-access handle. The function call will block until one or more bytes of the data has been written successfully, or until an error occurs.

Parameters

offset	The offset at which the data will be written.
buffers	One or more data buffers to be written to the handle.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes written. Returns 0 if an error occurred.

Remarks

The write_some operation may not transmit all of the data to the peer. Consider using the `write_at` function if you need to ensure that all data is written before the blocking operation completes.

windows::basic_stream_handle

Provides stream-oriented handle functionality.

```
template<
    typename StreamHandleService = stream_handle_service>
class basic_stream_handle :
  public windows::basic_handle< StreamHandleService >
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_handle is always the lowest layer.
native_handle_type	The native representation of a handle.
native_type	(Deprecated: Use native_handle_type.) The native representation of a handle.
service_type	The type of the service that will be used to provide I/O operations.

Member Functions

Name	Description
assign	Assign an existing native handle to the handle.
async_read_some	Start an asynchronous read.
async_write_some	Start an asynchronous write.
basic_stream_handle	Construct a basic_stream_handle without opening it. Construct a basic_stream_handle on an existing native handle. Move-construct a basic_stream_handle from another.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close the handle.
get_io_service	Get the io_service associated with the object.
is_open	Determine whether the handle is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native handle representation.
native_handle	Get the native handle representation.
operator=	Move-assign a basic_stream_handle from another.
read_some	Read some data from the handle.
write_some	Write some data to the handle.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `windows::basic_stream_handle` class template provides asynchronous and blocking stream-oriented handle functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/windows/basic_stream_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_stream_handle::assign

Assign an existing native handle to the handle.

```
void assign(
    const native_handle_type & handle);
  » more...

boost::system::error_code assign(
    const native_handle_type & handle,
    boost::system::error_code & ec);
  » more...
```

windows::basic_stream_handle::assign (1 of 2 overloads)

Inherited from windows::basic_handle.

Assign an existing native handle to the handle.

```
void assign(
    const native_handle_type & handle);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_stream_handle::assign (2 of 2 overloads)

Inherited from windows::basic_handle.

Assign an existing native handle to the handle.

```
boost::system::error_code assign(
    const native_handle_type & handle,
    boost::system::error_code & ec);
```

windows::basic_stream_handle::async_read_some

Start an asynchronous read.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_some(
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

This function is used to asynchronously read data from the stream handle. The function call always returns immediately.

Parameters

buffers — One or more buffers into which the data will be read. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the read operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes read.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The read operation may not read all of the requested number of bytes. Consider using the `async_read` function if you need to ensure that the requested amount of data is read before the asynchronous operation completes.

Example

To read into a single data buffer use the `buffer` function as follows:

```
handle.async_read_some(boost::asio::buffer(data, size), handler);
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_stream_handle::async_write_some

Start an asynchronous write.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_some(
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

This function is used to asynchronously write data to the stream handle. The function call always returns immediately.

Parameters

buffers — One or more data buffers to be written to the handle. Although the buffers object may be copied as necessary, ownership of the underlying memory blocks is retained by the caller, which must guarantee that they remain valid until the handler is called.

handler — The handler to be called when the write operation completes. Copies will be made of the handler as required. The function signature of the handler must be:

```
void handler(
  const boost::system::error_code& error, // Result of operation.
  std::size_t bytes_transferred           // Number of bytes written.
);
```

Regardless of whether the asynchronous operation completes immediately or not, the handler will not be invoked from within this function. Invocation of the handler will be performed in a manner equivalent to using `boost::asio::io_service::post()`.

Remarks

The write operation may not transmit all of the data to the peer. Consider using the `async_write` function if you need to ensure that all data is written before the asynchronous operation completes.

Example

To write a single data buffer use the `buffer` function as follows:

```
handle.async_write_some(boost::asio::buffer(data, size), handler);
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

windows::basic_stream_handle::basic_stream_handle

Construct a `windows::basic_stream_handle` without opening it.

```
explicit basic_stream_handle(
    boost::asio::io_service & io_service);
  » more...
```

Construct a `windows::basic_stream_handle` on an existing native handle.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
basic_stream_handle(
    boost::asio::io_service & io_service,
    const native_handle_type & handle);
  » more...
```

Move-construct a `windows::basic_stream_handle` from another.

```
basic_stream_handle(
    basic_stream_handle && other);
  » more...
```

windows::basic_stream_handle::basic_stream_handle (1 of 3 overloads)

Construct a `windows::basic_stream_handle` without opening it.

```
basic_stream_handle(
    boost::asio::io_service & io_service);
```

This constructor creates a stream handle without opening it. The handle needs to be opened and then connected or accepted before data can be sent or received on it.

Parameters

io_service — The `io_service` object that the stream handle will use to dispatch handlers for any asynchronous operations performed on the handle.

windows::basic_stream_handle::basic_stream_handle (2 of 3 overloads)

Construct a `windows::basic_stream_handle` on an existing native handle.

```
basic_stream_handle(
    boost::asio::io_service & io_service,
    const native_handle_type & handle);
```

This constructor creates a stream handle object to hold an existing native handle.

Parameters

io_service — The `io_service` object that the stream handle will use to dispatch handlers for any asynchronous operations performed on the handle.

handle — The new underlying handle implementation.

Exceptions

boost::system::system_error — Thrown on failure.

windows::basic_stream_handle::basic_stream_handle (3 of 3 overloads)

Move-construct a `windows::basic_stream_handle` from another.

```
basic_stream_handle(
    basic_stream_handle && other);
```

This constructor moves a stream handle from one object to another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

other The other `windows::basic_stream_handle` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_stream_handle(io_service&) constructor`.

windows::basic_stream_handle::cancel

Cancel all asynchronous operations associated with the handle.

```
void cancel();
  » more...

boost::system::error_code cancel(
    boost::system::error_code & ec);
  » more...
```

windows::basic_stream_handle::cancel (1 of 2 overloads)

Inherited from windows::basic_handle.

Cancel all asynchronous operations associated with the handle.

```
void cancel();
```

This function causes all outstanding asynchronous read or write operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error Thrown on failure.

windows::basic_stream_handle::cancel (2 of 2 overloads)

Inherited from windows::basic_handle.

Cancel all asynchronous operations associated with the handle.

```
boost::system::error_code cancel(
    boost::system::error_code & ec);
```

This function causes all outstanding asynchronous read or write operations to finish immediately, and the handlers for cancelled operations will be passed the `boost::asio::error::operation_aborted` error.

Parameters

ec Set to indicate what error occurred, if any.

windows::basic_stream_handle::close

Close the handle.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void close();
  » more...

boost::system::error_code close(
    boost::system::error_code & ec);
  » more...
```

windows::basic_stream_handle::close (1 of 2 overloads)

Inherited from windows::basic_handle.

Close the handle.

```
void close();
```

This function is used to close the handle. Any asynchronous read or write operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Exceptions

boost::system::system_error	Thrown on failure.

windows::basic_stream_handle::close (2 of 2 overloads)

Inherited from windows::basic_handle.

Close the handle.

```
boost::system::error_code close(
    boost::system::error_code & ec);
```

This function is used to close the handle. Any asynchronous read or write operations will be cancelled immediately, and will complete with the `boost::asio::error::operation_aborted` error.

Parameters

ec	Set to indicate what error occurred, if any.

windows::basic_stream_handle::get_implementation

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
  » more...

const implementation_type & get_implementation() const;
  » more...
```

windows::basic_stream_handle::get_implementation (1 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
implementation_type & get_implementation();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_stream_handle::get_implementation (2 of 2 overloads)

Inherited from basic_io_object.

Get the underlying implementation of the I/O object.

```
const implementation_type & get_implementation() const;
```

windows::basic_stream_handle::get_io_service

Inherited from basic_io_object.

Get the `io_service` associated with the object.

```
boost::asio::io_service & get_io_service();
```

This function may be used to obtain the `io_service` object that the I/O object uses to dispatch handlers for asynchronous operations.

Return Value

A reference to the `io_service` object that the I/O object will use to dispatch handlers. Ownership is not transferred to the caller.

windows::basic_stream_handle::get_service

Get the service associated with the I/O object.

```
service_type & get_service();
  » more...

const service_type & get_service() const;
  » more...
```

windows::basic_stream_handle::get_service (1 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
service_type & get_service();
```

windows::basic_stream_handle::get_service (2 of 2 overloads)

Inherited from basic_io_object.

Get the service associated with the I/O object.

```
const service_type & get_service() const;
```

windows::basic_stream_handle::implementation

Inherited from basic_io_object.

(Deprecated: Use `get_implementation()`.) The underlying implementation of the I/O object.

```
implementation_type implementation;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_stream_handle::implementation_type

Inherited from basic_io_object.

The underlying implementation type of I/O object.

```
typedef service_type::implementation_type implementation_type;
```

Requirements

Header: `boost/asio/windows/basic_stream_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_stream_handle::is_open

Inherited from windows::basic_handle.

Determine whether the handle is open.

```
bool is_open() const;
```

windows::basic_stream_handle::lowest_layer

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
  » more...
```

Get a const reference to the lowest layer.

```
const lowest_layer_type & lowest_layer() const;
  » more...
```

windows::basic_stream_handle::lowest_layer (1 of 2 overloads)

Inherited from windows::basic_handle.

Get a reference to the lowest layer.

```
lowest_layer_type & lowest_layer();
```

This function returns a reference to the lowest layer in a stack of layers. Since a `windows::basic_handle` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

windows::basic_stream_handle::lowest_layer (2 of 2 overloads)

Inherited from windows::basic_handle.

Get a const reference to the lowest layer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
const lowest_layer_type & lowest_layer() const;
```

This function returns a const reference to the lowest layer in a stack of layers. Since a `windows::basic_handle` cannot contain any further layers, it simply returns a reference to itself.

Return Value

A const reference to the lowest layer in the stack of layers. Ownership is not transferred to the caller.

windows::basic_stream_handle::lowest_layer_type

Inherited from windows::basic_handle.

A `windows::basic_handle` is always the lowest layer.

```
typedef basic_handle< StreamHandleService > lowest_layer_type;
```

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_handle is always the lowest layer.
native_handle_type	The native representation of a handle.
native_type	(Deprecated: Use native_handle_type.) The native representation of a handle.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native handle to the handle.
basic_handle	Construct a basic_handle without opening it. Construct a basic_handle on an existing native handle. Move-construct a basic_handle from another.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close the handle.
get_io_service	Get the io_service associated with the object.
is_open	Determine whether the handle is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native handle representation.
native_handle	Get the native handle representation.
operator=	Move-assign a basic_handle from another.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.
~basic_handle	Protected destructor to prevent deletion through this type.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `windows::basic_handle` class template provides the ability to wrap a Windows handle.

Thread Safety

Distinct objects: Safe.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Shared objects: Unsafe.

Requirements

Header: `boost/asio/windows/basic_stream_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_stream_handle::native

Inherited from windows::basic_handle.

(Deprecated: Use `native_handle()`.) Get the native handle representation.

```
native_type native();
```

This function may be used to obtain the underlying representation of the handle. This is intended to allow access to native handle functionality that is not otherwise provided.

windows::basic_stream_handle::native_handle

Inherited from windows::basic_handle.

Get the native handle representation.

```
native_handle_type native_handle();
```

This function may be used to obtain the underlying representation of the handle. This is intended to allow access to native handle functionality that is not otherwise provided.

windows::basic_stream_handle::native_handle_type

The native representation of a handle.

```
typedef StreamHandleService::native_handle_type native_handle_type;
```

Requirements

Header: `boost/asio/windows/basic_stream_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_stream_handle::native_type

(Deprecated: Use native_handle_type.) The native representation of a handle.

```
typedef StreamHandleService::native_handle_type native_type;
```

Requirements

Header: `boost/asio/windows/basic_stream_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_stream_handle::operator=

Move-assign a `windows::basic_stream_handle` from another.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
basic_stream_handle & operator=(
    basic_stream_handle && other);
```

This assignment operator moves a stream handle from one object to another.

Parameters

other The other `windows::basic_stream_handle` object from which the move will occur.

Remarks

Following the move, the moved-from object is in the same state as if constructed using the `basic_stream_handle(io_service&)` constructor.

windows::basic_stream_handle::read_some

Read some data from the handle.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
  » more...

template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

windows::basic_stream_handle::read_some (1 of 2 overloads)

Read some data from the handle.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers);
```

This function is used to read data from the stream handle. The function call will block until one or more bytes of data has been read successfully, or until an error occurs.

Parameters

buffers One or more buffers into which the data will be read.

Return Value

The number of bytes read.

Exceptions

boost::system::system_error	Thrown on failure. An error code of `boost::asio::error::eof` indicates that the connection was closed by the peer.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

The read_some operation may not read all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

Example

To read into a single data buffer use the `buffer` function as follows:

```
handle.read_some(boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on reading into multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

windows::basic_stream_handle::read_some (2 of 2 overloads)

Read some data from the handle.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to read data from the stream handle. The function call will block until one or more bytes of data has been read successfully, or until an error occurs.

Parameters

buffers	One or more buffers into which the data will be read.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes read. Returns 0 if an error occurred.

Remarks

The read_some operation may not read all of the requested number of bytes. Consider using the `read` function if you need to ensure that the requested amount of data is read before the blocking operation completes.

windows::basic_stream_handle::service

Inherited from basic_io_object.

(Deprecated: Use `get_service()`.) The service associated with the I/O object.

```
service_type & service;
```

Remarks

Available only for services that do not support movability.

windows::basic_stream_handle::service_type

Inherited from basic_io_object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The type of the service that will be used to provide I/O operations.

```
typedef StreamHandleService service_type;
```

Requirements

Header: `boost/asio/windows/basic_stream_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::basic_stream_handle::write_some

Write some data to the handle.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
  » more...

template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...
```

windows::basic_stream_handle::write_some (1 of 2 overloads)

Write some data to the handle.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers);
```

This function is used to write data to the stream handle. The function call will block until one or more bytes of the data has been written successfully, or until an error occurs.

Parameters

buffers	One or more data buffers to be written to the handle.

Return Value

The number of bytes written.

Exceptions

boost::system::system_error	Thrown on failure. An error code of `boost::asio::error::eof` indicates that the connection was closed by the peer.

Remarks

The write_some operation may not transmit all of the data to the peer. Consider using the `write` function if you need to ensure that all data is written before the blocking operation completes.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

To write a single data buffer use the `buffer` function as follows:

```
handle.write_some(boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

windows::basic_stream_handle::write_some (2 of 2 overloads)

Write some data to the handle.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to write data to the stream handle. The function call will block until one or more bytes of the data has been written successfully, or until an error occurs.

Parameters

buffers	One or more data buffers to be written to the handle.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes written. Returns 0 if an error occurred.

Remarks

The write_some operation may not transmit all of the data to the peer. Consider using the `write` function if you need to ensure that all data is written before the blocking operation completes.

windows::object_handle

Typedef for the typical usage of an object handle.

```
typedef basic_object_handle object_handle;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_handle is always the lowest layer.
native_handle_type	The native representation of a handle.
native_type	(Deprecated: Use native_handle_type.) The native representation of a handle.
service_type	The type of the service that will be used to provide I/O operations.

Member Functions

Name	Description
assign	Assign an existing native handle to the handle.
async_wait	Start an asynchronous wait on the object handle.
basic_object_handle	Construct a basic_object_handle without opening it. Construct a basic_object_handle on an existing native handle. Move-construct a basic_object_handle from another.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close the handle.
get_io_service	Get the io_service associated with the object.
is_open	Determine whether the handle is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native handle representation.
native_handle	Get the native handle representation.
operator=	Move-assign a basic_object_handle from another.
wait	Perform a blocking wait on the object handle.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `windows::basic_object_handle` class template provides asynchronous and blocking object-oriented handle functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/windows/object_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::object_handle_service

Default service implementation for an object handle.

```
class object_handle_service :
  public io_service::service
```

Types

Name	Description
implementation_type	The type of an object handle implementation.
native_handle_type	The native handle type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native handle to an object handle.
async_wait	Start an asynchronous wait.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close an object handle implementation.
construct	Construct a new object handle implementation.
destroy	Destroy an object handle implementation.
get_io_service	Get the io_service object that owns the service.
is_open	Determine whether the handle is open.
move_assign	Move-assign from another object handle implementation.
move_construct	Move-construct a new object handle implementation.
native_handle	Get the native handle implementation.
object_handle_service	Construct a new object handle service for the specified io_service.
wait	

Data Members

Name	Description
id	The unique service identifier.

Requirements

Header: `boost/asio/windows/object_handle_service.hpp`

Convenience header: `boost/asio.hpp`

windows::object_handle_service::assign

Assign an existing native handle to an object handle.

```
boost::system::error_code assign(
    implementation_type & impl,
    const native_handle_type & handle,
    boost::system::error_code & ec);
```

windows::object_handle_service::async_wait

Start an asynchronous wait.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename WaitHandler>
void async_wait(
    implementation_type & impl,
    WaitHandler handler);
```

windows::object_handle_service::cancel

Cancel all asynchronous operations associated with the handle.

```
boost::system::error_code cancel(
    implementation_type & impl,
    boost::system::error_code & ec);
```

windows::object_handle_service::close

Close an object handle implementation.

```
boost::system::error_code close(
    implementation_type & impl,
    boost::system::error_code & ec);
```

windows::object_handle_service::construct

Construct a new object handle implementation.

```
void construct(
    implementation_type & impl);
```

windows::object_handle_service::destroy

Destroy an object handle implementation.

```
void destroy(
    implementation_type & impl);
```

windows::object_handle_service::get_io_service

Inherited from io_service.

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

windows::object_handle_service::id

The unique service identifier.

```
static boost::asio::io_service::id id;
```

windows::object_handle_service::implementation_type

The type of an object handle implementation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
typedef implementation_defined implementation_type;
```

Requirements

Header: `boost/asio/windows/object_handle_service.hpp`

Convenience header: `boost/asio.hpp`

windows::object_handle_service::is_open

Determine whether the handle is open.

```
bool is_open(
    const implementation_type & impl) const;
```

windows::object_handle_service::move_assign

Move-assign from another object handle implementation.

```
void move_assign(
    implementation_type & impl,
    object_handle_service & other_service,
    implementation_type & other_impl);
```

windows::object_handle_service::move_construct

Move-construct a new object handle implementation.

```
void move_construct(
    implementation_type & impl,
    implementation_type & other_impl);
```

windows::object_handle_service::native_handle

Get the native handle implementation.

```
native_handle_type native_handle(
    implementation_type & impl);
```

windows::object_handle_service::native_handle_type

The native handle type.

```
typedef implementation_defined native_handle_type;
```

Requirements

Header: `boost/asio/windows/object_handle_service.hpp`

Convenience header: `boost/asio.hpp`

windows::object_handle_service::object_handle_service

Construct a new object handle service for the specified `io_service`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
object_handle_service(
    boost::asio::io_service & io_service);
```

windows::object_handle_service::wait

```
void wait(
    implementation_type & impl,
    boost::system::error_code & ec);
```

windows::overlapped_ptr

Wraps a handler to create an OVERLAPPED object for use with overlapped I/O.

```
class overlapped_ptr :
  noncopyable
```

Member Functions

Name	Description
complete	Post completion notification for overlapped operation. Releases ownership.
get	Get the contained OVERLAPPED object.
overlapped_ptr	Construct an empty overlapped_ptr. Construct an overlapped_ptr to contain the specified handler.
release	Release ownership of the OVERLAPPED object.
reset	Reset to empty. Reset to contain the specified handler, freeing any current OVERLAPPED object.
~overlapped_ptr	Destructor automatically frees the OVERLAPPED object unless released.

A special-purpose smart pointer used to wrap an application handler so that it can be passed as the LPOVERLAPPED argument to overlapped I/O functions.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/windows/overlapped_ptr.hpp`

Convenience header: `boost/asio.hpp`

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::overlapped_ptr::complete

Post completion notification for overlapped operation. Releases ownership.

```
void complete(
    const boost::system::error_code & ec,
    std::size_t bytes_transferred);
```

windows::overlapped_ptr::get

Get the contained OVERLAPPED object.

```
OVERLAPPED * get();
  » more...

const OVERLAPPED * get() const;
  » more...
```

windows::overlapped_ptr::get (1 of 2 overloads)

Get the contained OVERLAPPED object.

```
OVERLAPPED * get();
```

windows::overlapped_ptr::get (2 of 2 overloads)

Get the contained OVERLAPPED object.

```
const OVERLAPPED * get() const;
```

windows::overlapped_ptr::overlapped_ptr

Construct an empty `windows::overlapped_ptr`.

```
overlapped_ptr();
  » more...
```

Construct an `windows::overlapped_ptr` to contain the specified handler.

```
template<
    typename Handler>
explicit overlapped_ptr(
    boost::asio::io_service & io_service,
    Handler handler);
  » more...
```

windows::overlapped_ptr::overlapped_ptr (1 of 2 overloads)

Construct an empty `windows::overlapped_ptr`.

```
overlapped_ptr();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::overlapped_ptr::overlapped_ptr (2 of 2 overloads)

Construct an `windows::overlapped_ptr` to contain the specified handler.

```
template<
    typename Handler>
overlapped_ptr(
    boost::asio::io_service & io_service,
    Handler handler);
```

windows::overlapped_ptr::release

Release ownership of the OVERLAPPED object.

```
OVERLAPPED * release();
```

windows::overlapped_ptr::reset

Reset to empty.

```
void reset();
  » more...
```

Reset to contain the specified handler, freeing any current OVERLAPPED object.

```
template<
    typename Handler>
void reset(
    boost::asio::io_service & io_service,
    Handler handler);
  » more...
```

windows::overlapped_ptr::reset (1 of 2 overloads)

Reset to empty.

```
void reset();
```

windows::overlapped_ptr::reset (2 of 2 overloads)

Reset to contain the specified handler, freeing any current OVERLAPPED object.

```
template<
    typename Handler>
void reset(
    boost::asio::io_service & io_service,
    Handler handler);
```

windows::overlapped_ptr::~overlapped_ptr

Destructor automatically frees the OVERLAPPED object unless released.

```
~overlapped_ptr();
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::random_access_handle

Typedef for the typical usage of a random-access handle.

```
typedef basic_random_access_handle random_access_handle;
```

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_handle is always the lowest layer.
native_handle_type	The native representation of a handle.
native_type	(Deprecated: Use native_handle_type.) The native representation of a handle.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native handle to the handle.
async_read_some_at	Start an asynchronous read at the specified offset.
async_write_some_at	Start an asynchronous write at the specified offset.
basic_random_access_handle	Construct a basic_random_access_handle without opening it. Construct a basic_random_access_handle on an existing native handle. Move-construct a basic_random_access_handle from another.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close the handle.
get_io_service	Get the io_service associated with the object.
is_open	Determine whether the handle is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native handle representation.
native_handle	Get the native handle representation.
operator=	Move-assign a basic_random_access_handle from another.
read_some_at	Read some data from the handle at the specified offset.
write_some_at	Write some data to the handle at the specified offset.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

The `windows::basic_random_access_handle` class template provides asynchronous and blocking random-access handle functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/windows/random_access_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::random_access_handle_service

Default service implementation for a random-access handle.

```
class random_access_handle_service :
  public io_service::service
```

Types

Name	Description
implementation_type	The type of a random-access handle implementation.
native_handle_type	The native handle type.
native_type	(Deprecated: Use native_handle_type.) The native handle type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native handle to a random-access handle.
async_read_some_at	Start an asynchronous read at the specified offset.
async_write_some_at	Start an asynchronous write at the specified offset.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close a random-access handle implementation.
construct	Construct a new random-access handle implementation.
destroy	Destroy a random-access handle implementation.
get_io_service	Get the io_service object that owns the service.
is_open	Determine whether the handle is open.
move_assign	Move-assign from another random-access handle implementation.
move_construct	Move-construct a new random-access handle implementation.
native	(Deprecated: Use native_handle().) Get the native handle implementation.
native_handle	Get the native handle implementation.
random_access_handle_service	Construct a new random-access handle service for the specified io_service.
read_some_at	Read some data from the specified offset.
write_some_at	Write the given data at the specified offset.

Data Members

Name	Description
id	The unique service identifier.

Requirements

Header: `boost/asio/windows/random_access_handle_service.hpp`

Convenience header: `boost/asio.hpp`

windows::random_access_handle_service::assign

Assign an existing native handle to a random-access handle.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code assign(
    implementation_type & impl,
    const native_handle_type & handle,
    boost::system::error_code & ec);
```

windows::random_access_handle_service::async_read_some_at

Start an asynchronous read at the specified offset.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_some_at(
    implementation_type & impl,
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

windows::random_access_handle_service::async_write_some_at

Start an asynchronous write at the specified offset.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_some_at(
    implementation_type & impl,
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

windows::random_access_handle_service::cancel

Cancel all asynchronous operations associated with the handle.

```
boost::system::error_code cancel(
    implementation_type & impl,
    boost::system::error_code & ec);
```

windows::random_access_handle_service::close

Close a random-access handle implementation.

```
boost::system::error_code close(
    implementation_type & impl,
    boost::system::error_code & ec);
```

windows::random_access_handle_service::construct

Construct a new random-access handle implementation.

```
void construct(
    implementation_type & impl);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::random_access_handle_service::destroy

Destroy a random-access handle implementation.

```
void destroy(
    implementation_type & impl);
```

windows::random_access_handle_service::get_io_service

Inherited from io_service.

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

windows::random_access_handle_service::id

The unique service identifier.

```
static boost::asio::io_service::id id;
```

windows::random_access_handle_service::implementation_type

The type of a random-access handle implementation.

```
typedef implementation_defined implementation_type;
```

Requirements

Header: `boost/asio/windows/random_access_handle_service.hpp`

Convenience header: `boost/asio.hpp`

windows::random_access_handle_service::is_open

Determine whether the handle is open.

```
bool is_open(
    const implementation_type & impl) const;
```

windows::random_access_handle_service::move_assign

Move-assign from another random-access handle implementation.

```
void move_assign(
    implementation_type & impl,
    random_access_handle_service & other_service,
    implementation_type & other_impl);
```

windows::random_access_handle_service::move_construct

Move-construct a new random-access handle implementation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void move_construct(
    implementation_type & impl,
    implementation_type & other_impl);
```

windows::random_access_handle_service::native

(Deprecated: Use `native_handle()`.) Get the native handle implementation.

```
native_type native(
    implementation_type & impl);
```

windows::random_access_handle_service::native_handle

Get the native handle implementation.

```
native_handle_type native_handle(
    implementation_type & impl);
```

windows::random_access_handle_service::native_handle_type

The native handle type.

```
typedef implementation_defined native_handle_type;
```

Requirements

Header: `boost/asio/windows/random_access_handle_service.hpp`

Convenience header: `boost/asio.hpp`

windows::random_access_handle_service::native_type

(Deprecated: Use native_handle_type.) The native handle type.

```
typedef implementation_defined native_type;
```

Requirements

Header: `boost/asio/windows/random_access_handle_service.hpp`

Convenience header: `boost/asio.hpp`

windows::random_access_handle_service::random_access_handle_service

Construct a new random-access handle service for the specified `io_service`.

```
random_access_handle_service(
    boost::asio::io_service & io_service);
```

windows::random_access_handle_service::read_some_at

Read some data from the specified offset.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename MutableBufferSequence>
std::size_t read_some_at(
    implementation_type & impl,
    boost::uint64_t offset,
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

windows::random_access_handle_service::write_some_at

Write the given data at the specified offset.

```
template<
    typename ConstBufferSequence>
std::size_t write_some_at(
    implementation_type & impl,
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

windows::stream_handle

Typedef for the typical usage of a stream-oriented handle.

```
typedef basic_stream_handle stream_handle;
```

Types

Name	Description
implementation_type	The underlying implementation type of I/O object.
lowest_layer_type	A basic_handle is always the lowest layer.
native_handle_type	The native representation of a handle.
native_type	(Deprecated: Use native_handle_type.) The native representation of a handle.
service_type	The type of the service that will be used to provide I/O operations.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native handle to the handle.
async_read_some	Start an asynchronous read.
async_write_some	Start an asynchronous write.
basic_stream_handle	Construct a basic_stream_handle without opening it. Construct a basic_stream_handle on an existing native handle. Move-construct a basic_stream_handle from another.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close the handle.
get_io_service	Get the io_service associated with the object.
is_open	Determine whether the handle is open.
lowest_layer	Get a reference to the lowest layer. Get a const reference to the lowest layer.
native	(Deprecated: Use native_handle().) Get the native handle representation.
native_handle	Get the native handle representation.
operator=	Move-assign a basic_stream_handle from another.
read_some	Read some data from the handle.
write_some	Write some data to the handle.

Protected Member Functions

Name	Description
get_implementation	Get the underlying implementation of the I/O object.
get_service	Get the service associated with the I/O object.

Protected Data Members

Name	Description
implementation	(Deprecated: Use get_implementation().) The underlying implementation of the I/O object.
service	(Deprecated: Use get_service().) The service associated with the I/O object.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

The `windows::basic_stream_handle` class template provides asynchronous and blocking stream-oriented handle functionality.

Thread Safety

Distinct objects: Safe.

Shared objects: Unsafe.

Requirements

Header: `boost/asio/windows/stream_handle.hpp`

Convenience header: `boost/asio.hpp`

windows::stream_handle_service

Default service implementation for a stream handle.

```
class stream_handle_service :
  public io_service::service
```

Types

Name	Description
implementation_type	The type of a stream handle implementation.
native_handle_type	The native handle type.
native_type	(Deprecated: Use native_handle_type.) The native handle type.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Member Functions

Name	Description
assign	Assign an existing native handle to a stream handle.
async_read_some	Start an asynchronous read.
async_write_some	Start an asynchronous write.
cancel	Cancel all asynchronous operations associated with the handle.
close	Close a stream handle implementation.
construct	Construct a new stream handle implementation.
destroy	Destroy a stream handle implementation.
get_io_service	Get the io_service object that owns the service.
is_open	Determine whether the handle is open.
move_assign	Move-assign from another stream handle implementation.
move_construct	Move-construct a new stream handle implementation.
native	(Deprecated: Use native_handle().) Get the native handle implementation.
native_handle	Get the native handle implementation.
read_some	Read some data from the stream.
stream_handle_service	Construct a new stream handle service for the specified io_service.
write_some	Write the given data to the stream.

Data Members

Name	Description
id	The unique service identifier.

Requirements

Header: `boost/asio/windows/stream_handle_service.hpp`

Convenience header: `boost/asio.hpp`

windows::stream_handle_service::assign

Assign an existing native handle to a stream handle.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::system::error_code assign(
    implementation_type & impl,
    const native_handle_type & handle,
    boost::system::error_code & ec);
```

windows::stream_handle_service::async_read_some

Start an asynchronous read.

```
template<
    typename MutableBufferSequence,
    typename ReadHandler>
void async_read_some(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    ReadHandler handler);
```

windows::stream_handle_service::async_write_some

Start an asynchronous write.

```
template<
    typename ConstBufferSequence,
    typename WriteHandler>
void async_write_some(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    WriteHandler handler);
```

windows::stream_handle_service::cancel

Cancel all asynchronous operations associated with the handle.

```
boost::system::error_code cancel(
    implementation_type & impl,
    boost::system::error_code & ec);
```

windows::stream_handle_service::close

Close a stream handle implementation.

```
boost::system::error_code close(
    implementation_type & impl,
    boost::system::error_code & ec);
```

windows::stream_handle_service::construct

Construct a new stream handle implementation.

```
void construct(
    implementation_type & impl);
```

windows::stream_handle_service::destroy

Destroy a stream handle implementation.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
void destroy(
    implementation_type & impl);
```

windows::stream_handle_service::get_io_service

Inherited from io_service.

Get the `io_service` object that owns the service.

```
boost::asio::io_service & get_io_service();
```

windows::stream_handle_service::id

The unique service identifier.

```
static boost::asio::io_service::id id;
```

windows::stream_handle_service::implementation_type

The type of a stream handle implementation.

```
typedef implementation_defined implementation_type;
```

Requirements

Header: `boost/asio/windows/stream_handle_service.hpp`

Convenience header: `boost/asio.hpp`

windows::stream_handle_service::is_open

Determine whether the handle is open.

```
bool is_open(
    const implementation_type & impl) const;
```

windows::stream_handle_service::move_assign

Move-assign from another stream handle implementation.

```
void move_assign(
    implementation_type & impl,
    stream_handle_service & other_service,
    implementation_type & other_impl);
```

windows::stream_handle_service::move_construct

Move-construct a new stream handle implementation.

```
void move_construct(
    implementation_type & impl,
    implementation_type & other_impl);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::stream_handle_service::native

(Deprecated: Use `native_handle()`.) Get the native handle implementation.

```
native_type native(
    implementation_type & impl);
```

windows::stream_handle_service::native_handle

Get the native handle implementation.

```
native_handle_type native_handle(
    implementation_type & impl);
```

windows::stream_handle_service::native_handle_type

The native handle type.

```
typedef implementation_defined native_handle_type;
```

Requirements

Header: `boost/asio/windows/stream_handle_service.hpp`

Convenience header: `boost/asio.hpp`

windows::stream_handle_service::native_type

(Deprecated: Use native_handle_type.) The native handle type.

```
typedef implementation_defined native_type;
```

Requirements

Header: `boost/asio/windows/stream_handle_service.hpp`

Convenience header: `boost/asio.hpp`

windows::stream_handle_service::read_some

Read some data from the stream.

```
template<
    typename MutableBufferSequence>
std::size_t read_some(
    implementation_type & impl,
    const MutableBufferSequence & buffers,
    boost::system::error_code & ec);
```

windows::stream_handle_service::stream_handle_service

Construct a new stream handle service for the specified `io_service`.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
stream_handle_service(
    boost::asio::io_service & io_service);
```

windows::stream_handle_service::write_some

Write the given data to the stream.

```
template<
    typename ConstBufferSequence>
std::size_t write_some(
    implementation_type & impl,
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

write

Write a certain amount of data to a stream before returning.

```
template<
    typename SyncWriteStream,
    typename ConstBufferSequence>
std::size_t write(
    SyncWriteStream & s,
    const ConstBufferSequence & buffers);
  » more...

template<
    typename SyncWriteStream,
    typename ConstBufferSequence>
std::size_t write(
    SyncWriteStream & s,
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...

template<
    typename SyncWriteStream,
    typename ConstBufferSequence,
    typename CompletionCondition>
std::size_t write(
    SyncWriteStream & s,
    const ConstBufferSequence & buffers,
    CompletionCondition completion_condition);
  » more...

template<
    typename SyncWriteStream,
    typename ConstBufferSequence,
    typename CompletionCondition>
std::size_t write(
    SyncWriteStream & s,
    const ConstBufferSequence & buffers,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
  » more...

template<
    typename SyncWriteStream,
    typename Allocator>
std::size_t write(
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    SyncWriteStream & s,
    basic_streambuf< Allocator > & b);
  » more...

template<
    typename SyncWriteStream,
    typename Allocator>
std::size_t write(
    SyncWriteStream & s,
    basic_streambuf< Allocator > & b,
    boost::system::error_code & ec);
  » more...

template<
    typename SyncWriteStream,
    typename Allocator,
    typename CompletionCondition>
std::size_t write(
    SyncWriteStream & s,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition);
  » more...

template<
    typename SyncWriteStream,
    typename Allocator,
    typename CompletionCondition>
std::size_t write(
    SyncWriteStream & s,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
  » more...
```

Requirements

Header: `boost/asio/write.hpp`

Convenience header: `boost/asio.hpp`

write (1 of 8 overloads)

Write all of the supplied data to a stream before returning.

```
template<
    typename SyncWriteStream,
    typename ConstBufferSequence>
std::size_t write(
    SyncWriteStream & s,
    const ConstBufferSequence & buffers);
```

This function is used to write a certain number of bytes of data to a stream. The call will block until one of the following conditions is true:

- All of the data in the supplied buffers has been written. That is, the bytes transferred is equal to the sum of the buffer sizes.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's write_some function.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

s	The stream to which the data is to be written. The type must support the SyncWriteStream concept.
buffers	One or more buffers containing the data to be written. The sum of the buffer sizes indicates the maximum number of bytes to write to the stream.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error	Thrown on failure.

Example

To write a single data buffer use the `buffer` function as follows:

```
boost::asio::write(s, boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

Remarks

This overload is equivalent to calling:

```
boost::asio::write(
    s, buffers,
    boost::asio::transfer_all());
```

write (2 of 8 overloads)

Write all of the supplied data to a stream before returning.

```
template<
    typename SyncWriteStream,
    typename ConstBufferSequence>
std::size_t write(
    SyncWriteStream & s,
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to write a certain number of bytes of data to a stream. The call will block until one of the following conditions is true:

- All of the data in the supplied buffers has been written. That is, the bytes transferred is equal to the sum of the buffer sizes.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's write_some function.

Parameters

s	The stream to which the data is to be written. The type must support the SyncWriteStream concept.
buffers	One or more buffers containing the data to be written. The sum of the buffer sizes indicates the maximum number of bytes to write to the stream.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes transferred.

Example

To write a single data buffer use the `buffer` function as follows:

```
boost::asio::write(s, boost::asio::buffer(data, size), ec);
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

Remarks

This overload is equivalent to calling:

```
boost::asio::write(
    s, buffers,
    boost::asio::transfer_all(), ec);
```

write (3 of 8 overloads)

Write a certain amount of data to a stream before returning.

```
template<
    typename SyncWriteStream,
    typename ConstBufferSequence,
    typename CompletionCondition>
std::size_t write(
    SyncWriteStream & s,
    const ConstBufferSequence & buffers,
    CompletionCondition completion_condition);
```

This function is used to write a certain number of bytes of data to a stream. The call will block until one of the following conditions is true:

- All of the data in the supplied buffers has been written. That is, the bytes transferred is equal to the sum of the buffer sizes.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the stream's write_some function.

Parameters

s	The stream to which the data is to be written. The type must support the SyncWriteStream concept.
buffers	One or more buffers containing the data to be written. The sum of the buffer sizes indicates the maximum number of bytes to write to the stream.
completion_condition	The function object to be called to determine whether the write operation is complete. The signature of the function object must be:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t completion_condition(
  // Result of latest write_some operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the write operation is complete. A non-zero return value indicates the maximum number of bytes to be written on the next call to the stream's write_some function.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error	Thrown on failure.

Example

To write a single data buffer use the `buffer` function as follows:

```
boost::asio::write(s, boost::asio::buffer(data, size),
    boost::asio::transfer_at_least(32));
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

write (4 of 8 overloads)

Write a certain amount of data to a stream before returning.

```
template<
    typename SyncWriteStream,
    typename ConstBufferSequence,
    typename CompletionCondition>
std::size_t write(
    SyncWriteStream & s,
    const ConstBufferSequence & buffers,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
```

This function is used to write a certain number of bytes of data to a stream. The call will block until one of the following conditions is true:

- All of the data in the supplied buffers has been written. That is, the bytes transferred is equal to the sum of the buffer sizes.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the stream's write_some function.

Parameters

s	The stream to which the data is to be written. The type must support the SyncWriteStream concept.
buffers	One or more buffers containing the data to be written. The sum of the buffer sizes indicates the maximum number of bytes to write to the stream.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

completion_condition — The function object to be called to determine whether the write operation is complete. The signature of the function object must be:

```
std::size_t completion_condition(
  // Result of latest write_some operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the write operation is complete. A non-zero return value indicates the maximum number of bytes to be written on the next call to the stream's write_some function.

ec — Set to indicate what error occurred, if any.

Return Value

The number of bytes written. If an error occurs, returns the total number of bytes successfully transferred prior to the error.

write (5 of 8 overloads)

Write all of the supplied data to a stream before returning.

```
template<
    typename SyncWriteStream,
    typename Allocator>
std::size_t write(
    SyncWriteStream & s,
    basic_streambuf< Allocator > & b);
```

This function is used to write a certain number of bytes of data to a stream. The call will block until one of the following conditions is true:

- All of the data in the supplied `basic_streambuf` has been written.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's write_some function.

Parameters

s The stream to which the data is to be written. The type must support the SyncWriteStream concept.

b The `basic_streambuf` object from which data will be written.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error — Thrown on failure.

Remarks

This overload is equivalent to calling:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
boost::asio::write(
    s, b,
    boost::asio::transfer_all());
```

write (6 of 8 overloads)

Write all of the supplied data to a stream before returning.

```
template<
    typename SyncWriteStream,
    typename Allocator>
std::size_t write(
    SyncWriteStream & s,
    basic_streambuf< Allocator > & b,
    boost::system::error_code & ec);
```

This function is used to write a certain number of bytes of data to a stream. The call will block until one of the following conditions is true:

- All of the data in the supplied `basic_streambuf` has been written.
- An error occurred.

This operation is implemented in terms of zero or more calls to the stream's write_some function.

Parameters

s The stream to which the data is to be written. The type must support the SyncWriteStream concept.

b The `basic_streambuf` object from which data will be written.

ec Set to indicate what error occurred, if any.

Return Value

The number of bytes transferred.

Remarks

This overload is equivalent to calling:

```
boost::asio::write(
    s, b,
    boost::asio::transfer_all(), ec);
```

write (7 of 8 overloads)

Write a certain amount of data to a stream before returning.

```
template<
    typename SyncWriteStream,
    typename Allocator,
    typename CompletionCondition>
std::size_t write(
    SyncWriteStream & s,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition);
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

This function is used to write a certain number of bytes of data to a stream. The call will block until one of the following conditions is true:

- All of the data in the supplied `basic_streambuf` has been written.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the stream's write_some function.

Parameters

s	The stream to which the data is to be written. The type must support the SyncWriteStream concept.
b	The `basic_streambuf` object from which data will be written.

completion_condition	The function object to be called to determine whether the write operation is complete. The signature of the function object must be:
	
	`std::size_t completion_condition(`
	`  // Result of latest write_some operation.`
	`  const boost::system::error_code& error,`
	
	`  // Number of bytes transferred so far.`
	`  std::size_t bytes_transferred`
	`);`
	
	A return value of 0 indicates that the write operation is complete. A non-zero return value indicates the maximum number of bytes to be written on the next call to the stream's write_some function.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error	Thrown on failure.

write (8 of 8 overloads)

Write a certain amount of data to a stream before returning.

```
template<
    typename SyncWriteStream,
    typename Allocator,
    typename CompletionCondition>
std::size_t write(
    SyncWriteStream & s,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
```

This function is used to write a certain number of bytes of data to a stream. The call will block until one of the following conditions is true:

- All of the data in the supplied `basic_streambuf` has been written.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the stream's write_some function.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Parameters

s	The stream to which the data is to be written. The type must support the SyncWriteStream concept.
b	The `basic_streambuf` object from which data will be written.

completion_condition	The function object to be called to determine whether the write operation is complete. The signature of the function object must be:
	
	`std::size_t completion_condition(`
	`  // Result of latest write_some operation.`
	`  const boost::system::error_code& error,`
	
	`  // Number of bytes transferred so far.`
	`  std::size_t bytes_transferred`
	`);`
	
	A return value of 0 indicates that the write operation is complete. A non-zero return value indicates the maximum number of bytes to be written on the next call to the stream's write_some function.

ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes written. If an error occurs, returns the total number of bytes successfully transferred prior to the error.

write_at

Write a certain amount of data at a specified offset before returning.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename SyncRandomAccessWriteDevice,
    typename ConstBufferSequence>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    const ConstBufferSequence & buffers);
  » more...

template<
    typename SyncRandomAccessWriteDevice,
    typename ConstBufferSequence>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
  » more...

template<
    typename SyncRandomAccessWriteDevice,
    typename ConstBufferSequence,
    typename CompletionCondition>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    CompletionCondition completion_condition);
  » more...

template<
    typename SyncRandomAccessWriteDevice,
    typename ConstBufferSequence,
    typename CompletionCondition>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
  » more...

template<
    typename SyncRandomAccessWriteDevice,
    typename Allocator>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b);
  » more...

template<
    typename SyncRandomAccessWriteDevice,
    typename Allocator>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    boost::system::error_code & ec);
  » more...

template<
    typename SyncRandomAccessWriteDevice,
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
    typename Allocator,
    typename CompletionCondition>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition);
  » more...

template<
    typename SyncRandomAccessWriteDevice,
    typename Allocator,
    typename CompletionCondition>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
  » more...
```

Requirements

Header: `boost/asio/write_at.hpp`

Convenience header: `boost/asio.hpp`

write_at (1 of 8 overloads)

Write all of the supplied data at the specified offset before returning.

```
template<
    typename SyncRandomAccessWriteDevice,
    typename ConstBufferSequence>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    const ConstBufferSequence & buffers);
```

This function is used to write a certain number of bytes of data to a random access device at a specified offset. The call will block until one of the following conditions is true:

- All of the data in the supplied buffers has been written. That is, the bytes transferred is equal to the sum of the buffer sizes.
- An error occurred.

This operation is implemented in terms of zero or more calls to the device's write_some_at function.

Parameters

d	The device to which the data is to be written. The type must support the SyncRandomAccessWriteDevice concept.
offset	The offset at which the data will be written.
buffers	One or more buffers containing the data to be written. The sum of the buffer sizes indicates the maximum number of bytes to write to the device.

Return Value

The number of bytes transferred.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Exceptions

boost::system::system_error	Thrown on failure.

Example

To write a single data buffer use the `buffer` function as follows:

```
boost::asio::write_at(d, 42, boost::asio::buffer(data, size));
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

Remarks

This overload is equivalent to calling:

```
boost::asio::write_at(
    d, offset, buffers,
    boost::asio::transfer_all());
```

write_at (2 of 8 overloads)

Write all of the supplied data at the specified offset before returning.

```
template<
    typename SyncRandomAccessWriteDevice,
    typename ConstBufferSequence>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    boost::system::error_code & ec);
```

This function is used to write a certain number of bytes of data to a random access device at a specified offset. The call will block until one of the following conditions is true:

- All of the data in the supplied buffers has been written. That is, the bytes transferred is equal to the sum of the buffer sizes.
- An error occurred.

This operation is implemented in terms of zero or more calls to the device's write_some_at function.

Parameters

d	The device to which the data is to be written. The type must support the SyncRandomAccessWriteDevice concept.
offset	The offset at which the data will be written.
buffers	One or more buffers containing the data to be written. The sum of the buffer sizes indicates the maximum number of bytes to write to the device.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes transferred.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Example

To write a single data buffer use the `buffer` function as follows:

```
boost::asio::write_at(d, 42,
    boost::asio::buffer(data, size), ec);
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

Remarks

This overload is equivalent to calling:

```
boost::asio::write_at(
    d, offset, buffers,
    boost::asio::transfer_all(), ec);
```

write_at (3 of 8 overloads)

Write a certain amount of data at a specified offset before returning.

```
template<
    typename SyncRandomAccessWriteDevice,
    typename ConstBufferSequence,
    typename CompletionCondition>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    CompletionCondition completion_condition);
```

This function is used to write a certain number of bytes of data to a random access device at a specified offset. The call will block until one of the following conditions is true:

- All of the data in the supplied buffers has been written. That is, the bytes transferred is equal to the sum of the buffer sizes.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the device's write_some_at function.

Parameters

d	The device to which the data is to be written. The type must support the SyncRandomAccessWriteDevice concept.
offset	The offset at which the data will be written.
buffers	One or more buffers containing the data to be written. The sum of the buffer sizes indicates the maximum number of bytes to write to the device.
completion_condition	The function object to be called to determine whether the write operation is complete. The signature of the function object must be:

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
std::size_t completion_condition(
  // Result of latest write_some_at operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the write operation is complete. A non-zero return value indicates the maximum number of bytes to be written on the next call to the device's write_some_at function.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error	Thrown on failure.

Example

To write a single data buffer use the `buffer` function as follows:

```
boost::asio::write_at(d, 42, boost::asio::buffer(data, size),
    boost::asio::transfer_at_least(32));
```

See the `buffer` documentation for information on writing multiple buffers in one go, and how to use it with arrays, boost::array or std::vector.

write_at (4 of 8 overloads)

Write a certain amount of data at a specified offset before returning.

```
template<
    typename SyncRandomAccessWriteDevice,
    typename ConstBufferSequence,
    typename CompletionCondition>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    const ConstBufferSequence & buffers,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
```

This function is used to write a certain number of bytes of data to a random access device at a specified offset. The call will block until one of the following conditions is true:

- All of the data in the supplied buffers has been written. That is, the bytes transferred is equal to the sum of the buffer sizes.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the device's write_some_at function.

Parameters

d	The device to which the data is to be written. The type must support the SyncRandomAccessWriteDevice concept.
offset	The offset at which the data will be written.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffers — One or more buffers containing the data to be written. The sum of the buffer sizes indicates the maximum number of bytes to write to the device.

completion_condition — The function object to be called to determine whether the write operation is complete. The signature of the function object must be:

```
std::size_t completion_condition(
  // Result of latest write_some_at operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the write operation is complete. A non-zero return value indicates the maximum number of bytes to be written on the next call to the device's write_some_at function.

ec — Set to indicate what error occurred, if any.

Return Value

The number of bytes written. If an error occurs, returns the total number of bytes successfully transferred prior to the error.

write_at (5 of 8 overloads)

Write all of the supplied data at the specified offset before returning.

```
template<
    typename SyncRandomAccessWriteDevice,
    typename Allocator>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b);
```

This function is used to write a certain number of bytes of data to a random access device at a specified offset. The call will block until one of the following conditions is true:

- All of the data in the supplied `basic_streambuf` has been written.
- An error occurred.

This operation is implemented in terms of zero or more calls to the device's write_some_at function.

Parameters

d — The device to which the data is to be written. The type must support the SyncRandomAccessWriteDevice concept.

offset — The offset at which the data will be written.

b — The `basic_streambuf` object from which data will be written.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error — Thrown on failure.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Remarks

This overload is equivalent to calling:

```
boost::asio::write_at(
    d, 42, b,
    boost::asio::transfer_all());
```

write_at (6 of 8 overloads)

Write all of the supplied data at the specified offset before returning.

```
template<
    typename SyncRandomAccessWriteDevice,
    typename Allocator>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    boost::system::error_code & ec);
```

This function is used to write a certain number of bytes of data to a random access device at a specified offset. The call will block until one of the following conditions is true:

- All of the data in the supplied `basic_streambuf` has been written.
- An error occurred.

This operation is implemented in terms of zero or more calls to the device's write_some_at function.

Parameters

d	The device to which the data is to be written. The type must support the SyncRandomAccessWriteDevice concept.
offset	The offset at which the data will be written.
b	The `basic_streambuf` object from which data will be written.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes transferred.

Remarks

This overload is equivalent to calling:

```
boost::asio::write_at(
    d, 42, b,
    boost::asio::transfer_all(), ec);
```

write_at (7 of 8 overloads)

Write a certain amount of data at a specified offset before returning.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename SyncRandomAccessWriteDevice,
    typename Allocator,
    typename CompletionCondition>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition);
```

This function is used to write a certain number of bytes of data to a random access device at a specified offset. The call will block until one of the following conditions is true:

- All of the data in the supplied `basic_streambuf` has been written.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the device's write_some_at function.

Parameters

d	The device to which the data is to be written. The type must support the SyncRandomAccessWriteDevice concept.
offset	The offset at which the data will be written.
b	The `basic_streambuf` object from which data will be written.
completion_condition	The function object to be called to determine whether the write operation is complete. The signature of the function object must be:

```
std::size_t completion_condition(
  // Result of latest write_some_at operation.
  const boost::system::error_code& error,

  // Number of bytes transferred so far.
  std::size_t bytes_transferred
);
```

A return value of 0 indicates that the write operation is complete. A non-zero return value indicates the maximum number of bytes to be written on the next call to the device's write_some_at function.

Return Value

The number of bytes transferred.

Exceptions

boost::system::system_error	Thrown on failure.

write_at (8 of 8 overloads)

Write a certain amount of data at a specified offset before returning.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

```
template<
    typename SyncRandomAccessWriteDevice,
    typename Allocator,
    typename CompletionCondition>
std::size_t write_at(
    SyncRandomAccessWriteDevice & d,
    boost::uint64_t offset,
    basic_streambuf< Allocator > & b,
    CompletionCondition completion_condition,
    boost::system::error_code & ec);
```

This function is used to write a certain number of bytes of data to a random access device at a specified offset. The call will block until one of the following conditions is true:

- All of the data in the supplied `basic_streambuf` has been written.
- The completion_condition function object returns 0.

This operation is implemented in terms of zero or more calls to the device's write_some_at function.

Parameters

d	The device to which the data is to be written. The type must support the SyncRandomAccessWrite-Device concept.
offset	The offset at which the data will be written.
b	The `basic_streambuf` object from which data will be written.
completion_condition	The function object to be called to determine whether the write operation is complete. The signature of the function object must be: `std::size_t completion_condition( // Result of latest write_some_at operation. const boost::system::error_code& error, // Number of bytes transferred so far. std::size_t bytes_transferred );` A return value of 0 indicates that the write operation is complete. A non-zero return value indicates the maximum number of bytes to be written on the next call to the device's write_some_at function.
ec	Set to indicate what error occurred, if any.

Return Value

The number of bytes written. If an error occurs, returns the total number of bytes successfully transferred prior to the error.

boost::system::is_error_code_enum< boost::asio::error::addrinfo_errors >

```
template<>
struct boost::system::is_error_code_enum< boost::asio::error::addrinfo_errors >
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Data Members

Name	Description
value	

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

boost::system::is_error_code_enum< boost::asio::error::addrinfo_errors >::value

```
static const bool value = true;
```

boost::system::is_error_code_enum< boost::asio::error::basic_errors >

```
template<>
struct boost::system::is_error_code_enum< boost::asio::error::basic_errors >
```

Data Members

Name	Description
value	

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

boost::system::is_error_code_enum< boost::asio::error::basic_errors >::value

```
static const bool value = true;
```

boost::system::is_error_code_enum< boost::asio::error::misc_errors >

```
template<>
struct boost::system::is_error_code_enum< boost::asio::error::misc_errors >
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Data Members

Name	Description
value	

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

boost::system::is_error_code_enum< boost::asio::error::misc_errors >::value

```
static const bool value = true;
```

boost::system::is_error_code_enum< boost::asio::error::netdb_errors >

```
template<>
struct boost::system::is_error_code_enum< boost::asio::error::netdb_errors >
```

Data Members

Name	Description
value	

Requirements

Header: `boost/asio/error.hpp`

Convenience header: `boost/asio.hpp`

boost::system::is_error_code_enum< boost::asio::error::netdb_errors >::value

```
static const bool value = true;
```

boost::system::is_error_code_enum< boost::asio::error::ssl_errors >

```
template<>
struct boost::system::is_error_code_enum< boost::asio::error::ssl_errors >
```

Data Members

Name	Description
value	

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Requirements

Header: `boost/asio/ssl/error.hpp`

Convenience header: `boost/asio/ssl.hpp`

boost::system::is_error_code_enum< boost::asio::error::ssl_errors >::value

```
static const bool value = true;
```


XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Revision History

Asio 1.8.0 / Boost 1.49

- Added a new class template `basic_waitable_timer` based around the C++11 clock type requirements. It may be used with the clocks from the C++11 `<chrono>` library facility or, if those are not available, Boost.Chrono. The typedefs `high_resolution_timer`, `steady_timer` and `system_timer` may be used to create timer objects for the standard clock types.
- Added a new `windows::object_handle` class for performing waits on Windows kernel objects. Thanks go to Boris Schaeling for contributing substantially to the development of this feature.
- On Linux, `connect()` can return EAGAIN in certain circumstances. Remapped this to another error so that it doesn't look like a non-blocking operation (#6048).
- Fixed a compile error on NetBSD (#6098).
- Fixed deadlock on Mac OS X (#6275).
- Fixed a regression in `buffered_write_stream` (#6310).
- Fixed a non-paged pool "leak" on Windows when an `io_service` is repeatedly run without anything to do (#6321).
- Reverted earlier change to allow some speculative operations to be performed without holding the lock, as it introduced a race condition in some multithreaded scenarios.
- Fixed a bug where the second buffer in an array of two buffers may be ignored if the first buffer is empty.

Asio 1.6.1 / Boost 1.48

- Implemented various performance improvements, including:
  - Using thread-local operation queues in single-threaded use cases (i.e. when `concurrency_hint` is 1) to eliminate a lock/unlock pair.
  - Allowing some `epoll_reactor` speculative operations to be performed without holding the lock.
  - Improving locality of reference by performing an `epoll_reactor`'s I/O operation immediately before the corresponding handler is called. This also improves scalability across CPUs when multiple threads are running the `io_service`.
  - Specialising asynchronous read and write operations for buffer sequences that are arrays (`boost::array` or `std::array`) of exactly two buffers.
- Fixed a compile error in the regex overload of `async_read_until` (#5688).
- Fixed a Windows-specific compile error by explicitly specifying the `signal()` function from the global namespace (#5722).
- Changed the `deadline_timer` implementation so that it does not read the clock unless the timer heap is non-empty.
- Changed the SSL stream's buffers' sizes so that they are large enough to hold a complete TLS record (#5854).
- Fixed the behaviour of the synchronous `null_buffers` operations so that they obey the user's non-blocking setting (#5756).
- Changed to set the size of the select `fd_set` at runtime when using Windows.
- Disabled an MSVC warning due to const qualifier being applied to function type.
- Fixed a crash that occurs when using the Intel C++ compiler (#5763).
- Changed the initialisation of the OpenSSL library so that it supports all available algorithms.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- Fixed the SSL error mapping used when the session is gracefully shut down.
- Added some latency test programs.
- Clarified that a read operation ends when the buffer is full (#5999).
- Fixed an exception safety issue in `epoll_reactor` initialisation (#6006).
- Made the number of strand implementations configurable by defining `BOOST_ASIO_STRAND_IMPLEMENTATIONS` to the desired number.
- Added support for a new `BOOST_ASIO_ENABLE_SEQUENTIAL_STRAND_ALLOCATION` flag which switches the allocation of strand implementations to use a round-robin approach rather than hashing.
- Fixed potential strand starvation issue that can occur when `strand.post()` is used.

Asio 1.6.0 / Boost 1.47

- Added support for signal handling, using a new class called `signal_set`. Programs may add one or more signals to the set, and then perform an `async_wait()` operation. The specified handler will be called when one of the signals occurs. The same signal number may be registered with multiple `signal_set` objects, however the signal number must be used only with Asio. Addresses #2879.
- Added handler tracking, a new debugging aid. When enabled by defining `BOOST_ASIO_ENABLE_HANDLER_TRACKING`, Asio writes debugging output to the standard error stream. The output records asynchronous operations and the relationships between their handlers. It may be post-processed using the included `handlerviz.pl` tool to create a visual representation of the handlers (requires GraphViz).
- Added support for timeouts on socket iostreams, such as `ip::tcp::iostream`. A timeout is set by calling `expires_at()` or `expires_from_now()` to establish a deadline. Any socket operations which occur past the deadline will put the iostream into a bad state.
- Added a new `error()` member function to socket iostreams, for retrieving the error code from the most recent system call.
- Added a new `basic_deadline_timer::cancel_one()` function. This function lets you cancel a single waiting handler on a timer. Handlers are cancelled in FIFO order.
- Added a new `transfer_exactly()` completion condition. This can be used to send or receive a specified number of bytes even if the total size of the buffer (or buffer sequence) is larger.
- Added new free functions `connect()` and `async_connect()`. These operations try each endpoint in a list until the socket is successfully connected, and are useful for creating TCP clients that work with both IPv4 and IPv6.
- Extended the `buffer_size()` function so that it works for buffer sequences in addition to individual buffers.
- Added a new `buffer_copy()` function that can be used to copy the raw bytes between individual buffers and buffer sequences.
- Added new non-throwing overloads of `read()`, `read_at()`, `write()` and `write_at()` that do not require a completion condition.
- Added friendlier compiler errors for when a completion handler does not meet the necessary type requirements. When C++0x is available (currently supported for `g++` 4.5 or later, and MSVC 10), `static_assert` is also used to generate an informative error message. This checking may be disabled by defining `BOOST_ASIO_DISABLE_HANDLER_TYPE_REQUIREMENTS`.
- Added a new, completely rewritten SSL implementation. The new implementation compiles faster, shows substantially improved performance, and supports custom memory allocation and handler invocation. It includes new API features such as certificate verification callbacks and has improved error reporting. The new implementation is source-compatible with the old for most uses. However, if necessary, the old implementation may still be used by defining `BOOST_ASIO_ENABLE_OLD_SSL`. Addresses #3702, #3958.
- Changed the separate compilation support such that, to use Asio's SSL capabilities, you should also include `boost/asio/ssl/impl/src.hpp` in one source file in your program.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- Changed the SSL implementation to support build environments where SSL v2 is explicitly disabled (#5453).
- Made the `is_loopback()`, `is_unspecified()` and `is_multicast()` functions consistently available across the `ip::address`, `ip::address_v4` and `ip::address_v6` classes (#3939).
- Added new `non_blocking()` functions for managing the non-blocking behaviour of a socket or descriptor. The `io_control()` commands named `non_blocking_io` are now deprecated in favour of these new functions.
- Added new `native_non_blocking()` functions for managing the non-blocking mode of the underlying socket or descriptor. These functions are intended to allow the encapsulation of arbitrary non-blocking system calls as asynchronous operations, in a way that is transparent to the user of the socket object. The functions have no effect on the behaviour of the synchronous operations of the socket or descriptor.
- Added the `io_control()` member function for socket acceptors (#3297).
- Added a `release()` member function to posix descriptors. This function releases ownership of the underlying native descriptor to the caller. Addresses #3900.
- Added support for sequenced packet sockets (`SOCK_SEQPACKET`).
- Added a new `io_service::stopped()` function that can be used to determine whether the `io_service` has stopped (i.e. a `reset()` call is needed prior to any further calls to `run()`, `run_one()`, `poll()` or `poll_one()`).
- For consistency with the C++0x standard library, deprecated the `native_type` typedefs in favour of `native_handle_type`, and the `native()` member functions in favour of `native_handle()`.
- Added support for C++0x move construction and assignment to sockets, serial ports, POSIX descriptors and Windows handles.
- Reduced the copying of handler function objects.
- Added support for C++0x move construction to further reduce (and in some cases eliminate) copying of handler objects.
- Added support for the `fork()` system call. Programs that use `fork()` must call `io_service.notify_fork()` at the appropriate times. Two new examples have been added showing how to use this feature. Addresses #3238, #4162.
- Cleaned up the handling of errors reported by the `close()` system call. In particular, assume that most operating systems won't have `close()` fail with `EWOULDBLOCK`, but if it does then set the blocking mode and restart the call. If any other error occurs, assume the descriptor is closed. Addresses #3307.
- Added new `asio::buffer()` overloads for `std::array`, when available.
- Changed the implementation to use the C++0x standard library templates `array`, `shared_ptr`, `weak_ptr` and `atomic` when they are available, rather than the Boost equivalents.
- Use C++0x variadic templates when available, rather than generating function overloads using Boost.Preprocessor.
- Changed exception reporting to include the function name in exception `what()` messages.
- Fixed insufficient initialisers warning with MinGW.
- Changed the `shutdown_service()` member functions to be private.
- Added archetypes for testing socket option functions.
- Changed the Boost.Asio examples so that they don't use Boost.Thread's convenience header. Use the header file that is specifically for the boost::thread class instead.
- Removed the dependency on OS-provided macros for the well-known IPv4 and IPv6 addresses. This should eliminate annoying "missing braces around initializer" warnings (#3741).
- Reduced the size of `ip::basic_endpoint<>` objects (such as `ip::tcp::endpoint` and `ip::udp::endpoint`).

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- Changed the reactor backends to assume that any descriptors or sockets added using `assign()` may have been `dup()`-ed, and so require explicit deregistration from the reactor (#4971).
- Removed the deprecated member functions named `io_service()`. The `get_io_service()` member functions should be used instead.
- Removed the deprecated typedefs `resolver_query` and `resolver_iterator` from the `ip::tcp`, `ip::udp` and `ip::icmp` classes.
- Modified the `buffers_iterator<>` and `ip::basic_resolver_iterator` classes so that the value_type typedefs are non-const byte types.
- Fixed warnings reported by g++'s `-Wshadow` compiler option (#3905).
- Added an explicit cast to convert the `FIONBIO` constant to int, to suppress a compiler warning on some platforms (#5128).
- Changed most examples to treat a failure by an accept operation as non-fatal (#5124).
- Fixed an error in the `tick_count_timer` example by making the duration type signed. Previously, a wait on an already-passed deadline would not return for a very long time (#5418).

Asio 1.4.9 / Boost 1.46.1

- `EV_ONESHOT` seems to cause problems on some versions of Mac OS X, with the `io_service` destructor getting stuck inside the `close()` system call. Changed the kqueue backend to use `EV_CLEAR` instead (#5021).
- Fixed compile failures with some versions of `g++` due to the use of anonymous enums (#4883).
- Fixed a bug on kqueue-based platforms, where some system calls that repeatedly fail with `EWOULDBLOCK` are not correctly re-registered with kqueue.
- Changed `asio::streambuf` to ensure that its internal pointers are updated correctly after the data has been modified using `std::streambuf` member functions.
- Fixed a bug that prevented the linger socket option from working on platforms other than Windows.

Asio 1.4.8 / Boost 1.46

- Fixed an integer overflow problem that occurs when `ip::address_v4::broadcast()` is used on 64-bit platforms.
- Fixed a problem on older Linux kernels (where epoll is used without timerfd support) that prevents timely delivery of deadline_timer handlers, after the program has been running for some time (#5045).

Asio 1.4.7 / Boost 1.45

- Fixed a problem on kqueue-based platforms where a `deadline_timer` may never fire if the `io_service` is running in a background thread (#4568).
- Fixed a const-correctness issue that prevented valid uses of `has_service<>` from compiling (#4638).
- Fixed MinGW cross-compilation (#4491).
- Removed dependency on deprecated Boost.System functions (#4672).
- Ensured `close()`/`closesocket()` failures are correctly propagated (#4573).
- Added a check for errors returned by `InitializeCriticalSectionAndSpinCount` (#4574).
- Added support for hardware flow control on QNX (#4625).
- Always use `pselect()` on HP-UX, if it is available (#4578).

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- Ensured handler arguments are passed as lvalues (#4744).
- Fixed Windows build when thread support is disabled (#4680).
- Fixed a Windows-specific problem where `deadline_timer` objects with expiry times set more than 5 minutes in the future may never expire (#4745).
- Fixed the `resolver` backend on BSD platforms so that an empty service name resolves to port number `0`, as per the documentation (#4690).
- Fixed read operations so that they do not accept buffer sequences of type `const_buffers_1` (#4746).
- Redefined `Protocol` and `id` to avoid clashing with Objective-C++ keywords (#4191).
- Fixed a `vector` reallocation performance issue that can occur when there are many active `deadline_timer` objects (#4780).
- Fixed the kqueue backend so that it compiles on NetBSD (#4662).
- Fixed the socket `io_control()` implementation on 64-bit Mac OS X and BSD platforms (#4782).
- Fixed a Windows-specific problem where failures from `accept()` are incorrectly treated as successes (#4859).
- Deprecated the separate compilation header `<boost/asio/impl/src.cpp>` in favour of `<boost/asio/impl/src.hpp>` (#4560).

Asio 1.4.6 / Boost 1.44

- Reduced compile times. (Note that some programs may need to add additional `#includes`, e.g. if the program uses `boost::array` but does not explicitly include `<boost/array.hpp>`.)
- Reduced the size of generated code.
- Refactored `deadline_timer` implementation to improve performance.
- Improved multiprocessor scalability on Windows by using a dedicated hidden thread to wait for timers.
- Improved performance of `asio::streambuf` with `async_read()` and `async_read_until()`. These read operations now use the existing capacity of the `streambuf` when reading, rather than limiting the read to 512 bytes.
- Added optional separate compilation. To enable, add `#include <boost/asio/impl/src.cpp>` to one source file in a program, then build the program with `BOOST_ASIO_SEPARATE_COMPILATION` defined in the project/compiler settings. Alternatively, `BOOST_ASIO_DYN_LINK` may be defined to build a separately-compiled Asio as part of a shared library.
- Added new macro `BOOST_ASIO_DISABLE_FENCED_BLOCK` to permit the disabling of memory fences around completion handlers, even if thread support is enabled.
- Reworked timeout examples to better illustrate typical use cases.
- Ensured that handler arguments are passed as `const` types.
- Fixed incorrect parameter order in `null_buffers` variant of `async_send_to` (#4170).
- Ensured `unsigned char` is used with `isdigit` in `getaddrinfo` emulation (#4201).
- Fixed handling of very small but non-zero timeouts (#4205).
- Fixed crash that occurred when an empty buffer sequence was passed to a composed read or write operation.
- Added missing `operator+` overload in `buffers_iterator` (#4382).
- Implemented cancellation of `null_buffers` operations on Windows.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Asio 1.4.5 / Boost 1.43

- Improved performance.
- Reduced compile times.
- Reduced the size of generated code.
- Extended the guarantee that background threads don't call user code to all asynchronous operations (#3923).
- Changed to use edge-triggered epoll on Linux.
- Changed to use `timerfd` for dispatching timers on Linux, when available.
- Changed to use one-shot notifications with kqueue on Mac OS X and BSD platforms.
- Added a bitmask type `ip::resolver_query_base::flags` as per the TR2 proposal. This type prevents implicit conversion from `int` to `flags`, allowing the compiler to catch cases where users incorrectly pass a numeric port number as the service name.
- Added `#define NOMINMAX` for all Windows compilers. Users can define `BOOST_ASIO_NO_NOMINMAX` to suppress this definition (#3901).
- Fixed a bug where 0-byte asynchronous reads were incorrectly passing an `error::eof` result to the completion handler (#4023).
- Changed the `io_control()` member functions to always call `ioctl` on the underlying descriptor when modifying blocking mode (#3307).
- Changed the resolver implementation to longer require the typedefs `InternetProtocol::resolver_query` and `InternetProtocol::resolver_iterator`, as neither typedef is part of the documented `InternetProtocol` requirements. The corresponding typedefs in the `ip::tcp`, `ip::udp` and `ip::icmp` classes have been deprecated.
- Fixed out-of-band handling for reactors not based on `select()`.
- Added new `BOOST_ASIO_DISABLE_THREADS` macro that allows Asio's threading support to be independently disabled.
- Minor documentation improvements.

Asio 1.4.4 / Boost 1.42

- Added a new HTTP Server 4 example illustrating the use of stackless coroutines with Asio.
- Changed handler allocation and invocation to use `boost::addressof` to get the address of handler objects, rather than applying `operator&` directly (#2977).
- Restricted MSVC buffer debugging workaround to 2008, as it causes a crash with 2010 beta 2 (#3796, #3822).
- Fixed a problem with the lifetime of handler memory, where Windows needs the `OVERLAPPED` structure to be valid until both the initiating function call has returned and the completion packet has been delivered.
- Don't block signals while performing system calls, but instead restart the calls if they are interrupted.
- Documented the guarantee made by strand objects with respect to order of handler invocation.
- Changed strands to use a pool of implementations, to make copying of strands cheaper.
- Ensured that kqueue support is enabled for BSD platforms (#3626).
- Added a `boost_` prefix to the `extern "C"` thread entry point function (#3809).
- In `getaddrinfo` emulation, only check the socket type (`SOCK_STREAM` or `SOCK_DGRAM`) if a service name has been specified. This should allow the emulation to work with raw sockets.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- Added a workaround for some broken Windows firewalls that make a socket appear bound to 0.0.0.0 when it is in fact bound to 127.0.0.1.
- Applied a fix for reported excessive CPU usage under Solaris (#3670).
- Added some support for platforms that use older compilers such as g++ 2.95 (#3743).

Asio 1.4.3 / Boost 1.40

- Added a new ping example to illustrate the use of ICMP sockets.
- Changed the `buffered*_stream<>` templates to treat 0-byte reads and writes as no-ops, to comply with the documented type requirements for `SyncReadStream`, `AsyncReadStream`, `SyncWriteStream` and `AsyncWriteStream`.
- Changed some instances of the `throw` keyword to `boost::throw_exception()` to allow Asio to be used when exception support is disabled. Note that the SSL wrappers still require exception support (#2754).
- Made Asio compatible with the OpenSSL 1.0 beta (#3256).
- Eliminated a redundant system call in the Solaris `/dev/poll` backend.
- Fixed a bug in resizing of the bucket array in the internal hash maps (#3095).
- Ensured correct propagation of the error code when a synchronous accept fails (#3216).
- Ensured correct propagation of the error code when a synchronous read or write on a Windows HANDLE fails.
- Fixed failures reported when `_GLIBCXX_DEBUG` is defined (#3098).
- Fixed custom memory allocation support for timers (#3107).
- Tidied up various warnings reported by g++ (#1341, #2618).
- Various documentation improvements, including more obvious hyperlinks to function overloads, header file information, examples for the handler type requirements, and adding enum values to the index (#3157, #2620).

Asio 1.4.2 / Boost 1.39

- Implement automatic resizing of the bucket array in the internal hash maps. This is to improve performance for very large numbers of asynchronous operations and also to reduce memory usage for very small numbers. A new macro `BOOST_ASIO_HASH_MAP_BUCKETS` may be used to tweak the sizes used for the bucket arrays. (N.B. this feature introduced a bug which was fixed in Asio 1.4.3 / Boost 1.40.)
- Add performance optimisation for the Windows IOCP backend for when no timers are used.
- Prevent locale settings from affecting formatting of TCP and UDP endpoints (#2682).
- Fix a memory leak that occurred when an asynchronous SSL operation's completion handler threw an exception (#2910).
- Fix the implementation of `io_control()` so that it adheres to the documented type requirements for IoControlCommand (#2820).
- Fix incompatibility between Asio and ncurses.h (#2156).
- On Windows, specifically handle the case when an overlapped `ReadFile` call fails with `ERROR_MORE_DATA`. This enables a hack where a `windows::stream_handle` can be used with a message-oriented named pipe (#2936).
- Fix system call wrappers to always clear the error on success, as POSIX allows successful system calls to modify errno (#2953).
- Don't include termios.h if `BOOST_ASIO_DISABLE_SERIAL_PORT` is defined (#2917).
- Cleaned up some more MSVC level 4 warnings (#2828).

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

- Various documentation fixes (#2871).

Asio 1.4.1 / Boost 1.38

- Improved compatibility with some Windows firewall software.
- Ensured arguments to `windows::overlapped_ptr::complete()` are correctly passed to the completion handler (#2614).
- Fixed a link problem and multicast failure on QNX (#2504, #2530).
- Fixed a compile error in SSL support on MinGW / g++ 3.4.5.
- Drop back to using a pipe for notification if eventfd is not available at runtime on Linux (#2683).
- Various minor bug and documentation fixes (#2534, #2541, #2607, #2617, #2619).

Asio 1.4.0 / Boost 1.37

- Enhanced CompletionCondition concept with the signature `size_t CompletionCondition(error_code ec, size_t total)`, where the return value indicates the maximum number of bytes to be transferred on the next read or write operation. (The old CompletionCondition signature is still supported for backwards compatibility).
- New windows::overlapped_ptr class to allow arbitrary overlapped I/O functions (such as TransmitFile) to be used with Asio.
- On recent versions of Linux, an eventfd descriptor is now used (rather than a pipe) to interrupt a blocked select/epoll reactor.
- Added const overloads of lowest_layer().
- Synchronous read, write, accept and connect operations are now thread safe (meaning that it is now permitted to perform concurrent synchronous operations on an individual socket, if supported by the OS).
- Reactor-based io_service implementations now use lazy initialisation to reduce the memory usage of an io_service object used only as a message queue.

Asio 1.2.0 / Boost 1.36

- Added support for serial ports.
- Added support for UNIX domain sockets.
- Added support for raw sockets and ICMP.
- Added wrappers for POSIX stream-oriented file descriptors (excluding regular files).
- Added wrappers for Windows stream-oriented `HANDLE`s such as named pipes (requires `HANDLE`s that work with I/O completion ports).
- Added wrappers for Windows random-access `HANDLE`s such as files (requires `HANDLE`s that work with I/O completion ports).
- Added support for reactor-style operations (i.e. they report readiness but perform no I/O) using a new `null_buffers` type.
- Added an iterator type for bytewise traversal of buffer sequences.
- Added new `read_until()` and `async_read_until()` overloads that take a user-defined function object for locating message boundaries.
- Added an experimental two-lock queue (enabled by defining `BOOST_ASIO_ENABLE_TWO_LOCK_QUEUE`) that may provide better `io_service` scalability across many processors.
- Various fixes, performance improvements, and more complete coverage of the custom memory allocation support.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Asio 1.0.0 / Boost 1.35

First release of Asio as part of Boost.

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

Index

Symbols

~basic_descriptor
posix::basic_descriptor, 970
~basic_handle
windows::basic_handle, 1170
~basic_io_object
basic_io_object, 287
~basic_socket
basic_socket, 491
~basic_socket_streambuf
basic_socket_streambuf, 584
~context
ssl::context, 1107
~context_base
ssl::context_base, 1110
~descriptor_base
posix::descriptor_base, 993
~io_service
io_service, 797
~overlapped_ptr
windows::overlapped_ptr, 1222
~resolver_query_base
ip::resolver_query_base, 880
~serial_port_base
serial_port_base, 1056
~service
io_service::service, 799
~socket_base
socket_base, 1089
~strand
io_service::strand, 803
~stream
ssl::stream, 1125
~stream_base
ssl::stream_base, 1127
~work
io_service::work, 805

A

accept
basic_socket_acceptor, 496
socket_acceptor_service, 1074
acceptor
ip::tcp, 888
local::stream_protocol, 932
access_denied
error::basic_errors, 775
add
basic_signal_set, 434
signal_set_service, 1070
time_traits< boost::posix_time::ptime >, 1150
address
ip::address, 807
ip::basic_endpoint, 833

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

address_configured
  ip::basic_resolver_query, 860
  ip::resolver_query_base, 879
address_family_not_supported
  error::basic_errors, 775
address_in_use
  error::basic_errors, 775
address_v4
  ip::address_v4, 815
address_v6
  ip::address_v6, 824
add_service, 162
  io_service, 788
add_verify_path
  ssl::context, 1091
all_matching
  ip::basic_resolver_query, 860
  ip::resolver_query_base, 879
already_connected
  error::basic_errors, 775
already_open
  error::misc_errors, 779
already_started
  error::basic_errors, 775
any
  ip::address_v4, 816
  ip::address_v6, 825
asio_handler_allocate, 163
asio_handler_deallocate, 164
asio_handler_invoke, 164
asn1
  ssl::context, 1093
  ssl::context_base, 1108
assign
  basic_datagram_socket, 208
  basic_raw_socket, 293
  basic_seq_packet_socket, 360
  basic_serial_port, 414
  basic_socket, 448
  basic_socket_acceptor, 498
  basic_socket_streambuf, 539
  basic_stream_socket, 590
  datagram_socket_service, 759
  posix::basic_descriptor, 955
  posix::basic_stream_descriptor, 972
  posix::stream_descriptor_service, 998
  raw_socket_service, 1006
  seq_packet_socket_service, 1045
  serial_port_service, 1063
  socket_acceptor_service, 1074
  stream_socket_service, 1137
  windows::basic_handle, 1161
  windows::basic_object_handle, 1172
  windows::basic_random_access_handle, 1184
  windows::basic_stream_handle, 1200
  windows::object_handle_service, 1217
  windows::random_access_handle_service, 1226
  windows::stream_handle_service, 1233

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

async_accept
  basic_socket_acceptor, 499
  socket_acceptor_service, 1074
async_connect, 165
  basic_datagram_socket, 209
  basic_raw_socket, 294
  basic_seq_packet_socket, 360
  basic_socket, 448
  basic_socket_streambuf, 540
  basic_stream_socket, 591
  datagram_socket_service, 759
  raw_socket_service, 1006
  seq_packet_socket_service, 1045
  stream_socket_service, 1137
async_fill
  buffered_read_stream, 707
  buffered_stream, 716
async_flush
  buffered_stream, 716
  buffered_write_stream, 725
async_handshake
  ssl::stream, 1114
async_read, 172
async_read_at, 178
async_read_some
  basic_serial_port, 414
  basic_stream_socket, 591
  buffered_read_stream, 707
  buffered_stream, 716
  buffered_write_stream, 725
  posix::basic_stream_descriptor, 973
  posix::stream_descriptor_service, 998
  serial_port_service, 1063
  ssl::stream, 1114
  windows::basic_stream_handle, 1201
  windows::stream_handle_service, 1234
async_read_some_at
  windows::basic_random_access_handle, 1185
  windows::random_access_handle_service, 1227
async_read_until, 184
async_receive
  basic_datagram_socket, 209
  basic_raw_socket, 294
  basic_seq_packet_socket, 361
  basic_stream_socket, 592
  datagram_socket_service, 760
  raw_socket_service, 1006
  seq_packet_socket_service, 1045
  stream_socket_service, 1137
async_receive_from
  basic_datagram_socket, 211
  basic_raw_socket, 296
  datagram_socket_service, 760
  raw_socket_service, 1007
async_resolve
  ip::basic_resolver, 840
  ip::resolver_service, 881
async_send

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_datagram_socket, 213
basic_raw_socket, 298
basic_seq_packet_socket, 363
basic_stream_socket, 594
datagram_socket_service, 760
raw_socket_service, 1007
seq_packet_socket_service, 1046
stream_socket_service, 1138
async_send_to
basic_datagram_socket, 215
basic_raw_socket, 300
datagram_socket_service, 760
raw_socket_service, 1007
async_shutdown
ssl::stream, 1115
async_wait
basic_deadline_timer, 273
basic_signal_set, 435
basic_waitable_timer, 657
deadline_timer_service, 771
signal_set_service, 1071
waitable_timer_service, 1155
windows::basic_object_handle, 1172
windows::object_handle_service, 1217
async_write, 192
async_write_at, 198
async_write_some
basic_serial_port, 415
basic_stream_socket, 596
buffered_read_stream, 707
buffered_stream, 716
buffered_write_stream, 725
posix::basic_stream_descriptor, 974
posix::stream_descriptor_service, 998
serial_port_service, 1063
ssl::stream, 1115
windows::basic_stream_handle, 1202
windows::stream_handle_service, 1234
async_write_some_at
windows::basic_random_access_handle, 1185
windows::random_access_handle_service, 1227
at_mark
basic_datagram_socket, 217
basic_raw_socket, 302
basic_seq_packet_socket, 364
basic_socket, 449
basic_socket_streambuf, 540
basic_stream_socket, 597
datagram_socket_service, 760
raw_socket_service, 1007
seq_packet_socket_service, 1046
stream_socket_service, 1138
available
basic_datagram_socket, 218
basic_raw_socket, 303
basic_seq_packet_socket, 365
basic_socket, 450
basic_socket_streambuf, 541

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_stream_socket, 598
datagram_socket_service, 761
raw_socket_service, 1008
seq_packet_socket_service, 1046
stream_socket_service, 1138

B

bad_descriptor
error::basic_errors, 775
basic_datagram_socket
basic_datagram_socket, 219
basic_deadline_timer
basic_deadline_timer, 273
basic_descriptor
posix::basic_descriptor, 955
basic_endpoint
ip::basic_endpoint, 834
local::basic_endpoint, 918
basic_handle
windows::basic_handle, 1161
basic_io_object
basic_io_object, 284
basic_object_handle
windows::basic_object_handle, 1173
basic_random_access_handle
windows::basic_random_access_handle, 1186
basic_raw_socket
basic_raw_socket, 304
basic_resolver
ip::basic_resolver, 842
basic_resolver_entry
ip::basic_resolver_entry, 851
basic_resolver_iterator
ip::basic_resolver_iterator, 854
basic_resolver_query
ip::basic_resolver_query, 860
basic_seq_packet_socket
basic_seq_packet_socket, 366
basic_serial_port
basic_serial_port, 416
basic_signal_set
basic_signal_set, 436
basic_socket
basic_socket, 451
basic_socket_acceptor
basic_socket_acceptor, 501
basic_socket_iostream
basic_socket_iostream, 530
basic_socket_streambuf
basic_socket_streambuf, 542
basic_streambuf
basic_streambuf, 651
basic_stream_descriptor
posix::basic_stream_descriptor, 974
basic_stream_handle
windows::basic_stream_handle, 1202
basic_stream_socket

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_stream_socket, 599
basic_waitable_timer
basic_waitable_timer, 658
baud_rate
serial_port_base::baud_rate, 1056
begin
buffers_iterator, 733
const_buffers_1, 753
mutable_buffers_1, 947
null_buffers, 950
bind
basic_datagram_socket, 221
basic_raw_socket, 306
basic_seq_packet_socket, 368
basic_socket, 453
basic_socket_acceptor, 504
basic_socket_streambuf, 542
basic_stream_socket, 601
datagram_socket_service, 761
raw_socket_service, 1008
seq_packet_socket_service, 1046
socket_acceptor_service, 1074
stream_socket_service, 1138
broadcast
basic_datagram_socket, 223
basic_raw_socket, 308
basic_seq_packet_socket, 369
basic_socket, 454
basic_socket_acceptor, 505
basic_socket_streambuf, 543
basic_stream_socket, 603
ip::address_v4, 816
socket_base, 1081
broken_pipe
error::basic_errors, 775
buffer, 668
buffered_read_stream
buffered_read_stream, 707
buffered_stream
buffered_stream, 717
buffered_write_stream
buffered_write_stream, 725
buffers_begin, 731
buffers_end, 731
buffers_iterator
buffers_iterator, 733
buffer_cast, 683
buffer_copy, 684
buffer_size, 703
bytes_readable
basic_datagram_socket, 223
basic_raw_socket, 308
basic_seq_packet_socket, 370
basic_socket, 455
basic_socket_acceptor, 505
basic_socket_streambuf, 544
basic_stream_socket, 603
posix::basic_descriptor, 957

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_stream_descriptor, 976
posix::descriptor_base, 993
socket_base, 1081
bytes_type
ip::address_v4, 817
ip::address_v6, 825

C

cancel
basic_datagram_socket, 224
basic_deadline_timer, 275
basic_raw_socket, 309
basic_seq_packet_socket, 370
basic_serial_port, 418
basic_signal_set, 438
basic_socket, 455
basic_socket_acceptor, 506
basic_socket_streambuf, 544
basic_stream_socket, 604
basic_waitable_timer, 659
datagram_socket_service, 761
deadline_timer_service, 771
ip::basic_resolver, 842
ip::resolver_service, 882
posix::basic_descriptor, 957
posix::basic_stream_descriptor, 976
posix::stream_descriptor_service, 998
raw_socket_service, 1008
seq_packet_socket_service, 1046
serial_port_service, 1063
signal_set_service, 1071
socket_acceptor_service, 1075
stream_socket_service, 1138
waitable_timer_service, 1155
windows::basic_handle, 1162
windows::basic_object_handle, 1174
windows::basic_random_access_handle, 1188
windows::basic_stream_handle, 1204
windows::object_handle_service, 1218
windows::random_access_handle_service, 1227
windows::stream_handle_service, 1234
cancel_one
basic_deadline_timer, 276
basic_waitable_timer, 660
deadline_timer_service, 772
waitable_timer_service, 1156
canonical_name
ip::basic_resolver_query, 863
ip::resolver_query_base, 879
capacity
ip::basic_endpoint, 835
local::basic_endpoint, 919
character_size
serial_port_base::character_size, 1057
clear
basic_signal_set, 439
signal_set_service, 1071

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

client
ssl::stream, 1117
ssl::stream_base, 1126
clock_type
basic_waitable_timer, 661
waitable_timer_service, 1156
close
basic_datagram_socket, 225
basic_raw_socket, 310
basic_seq_packet_socket, 372
basic_serial_port, 419
basic_socket, 456
basic_socket_acceptor, 506
basic_socket_iostream, 530
basic_socket_streambuf, 546
basic_stream_socket, 605
buffered_read_stream, 708
buffered_stream, 717
buffered_write_stream, 726
datagram_socket_service, 761
posix::basic_descriptor, 958
posix::basic_stream_descriptor, 977
posix::stream_descriptor_service, 999
raw_socket_service, 1008
seq_packet_socket_service, 1047
serial_port_service, 1064
socket_acceptor_service, 1075
stream_socket_service, 1139
windows::basic_handle, 1163
windows::basic_object_handle, 1175
windows::basic_random_access_handle, 1188
windows::basic_stream_handle, 1204
windows::object_handle_service, 1218
windows::random_access_handle_service, 1227
windows::stream_handle_service, 1234
commit
basic_streambuf, 651
complete
windows::overlapped_ptr, 1221
connect, 739
basic_datagram_socket, 226
basic_raw_socket, 311
basic_seq_packet_socket, 373
basic_socket, 457
basic_socket_iostream, 530
basic_socket_streambuf, 547
basic_stream_socket, 606
datagram_socket_service, 761
raw_socket_service, 1008
seq_packet_socket_service, 1047
stream_socket_service, 1139
connection_aborted
error::basic_errors, 775
connection_refused
error::basic_errors, 775
connection_reset
error::basic_errors, 775
construct

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

datagram_socket_service, 761
deadline_timer_service, 772
ip::resolver_service, 882
posix::stream_descriptor_service, 999
raw_socket_service, 1008
seq_packet_socket_service, 1047
serial_port_service, 1064
signal_set_service, 1071
socket_acceptor_service, 1075
stream_socket_service, 1139
waitable_timer_service, 1156
windows::object_handle_service, 1218
windows::random_access_handle_service, 1227
windows::stream_handle_service, 1234
const_buffer
const_buffer, 751
const_buffers_1
const_buffers_1, 753
const_buffers_type
basic_streambuf, 652
const_iterator
const_buffers_1, 753
mutable_buffers_1, 947
null_buffers, 950
consume
basic_streambuf, 652
context
ssl::context, 1092
create
ip::basic_resolver_iterator, 854

D

data
basic_streambuf, 652
ip::basic_endpoint, 835
local::basic_endpoint, 919
datagram_socket_service
datagram_socket_service, 762
data_type
ip::basic_endpoint, 835
local::basic_endpoint, 919
deadline_timer, 767
deadline_timer_service
deadline_timer_service, 772
debug
basic_datagram_socket, 228
basic_raw_socket, 313
basic_seq_packet_socket, 374
basic_socket, 459
basic_socket_acceptor, 507
basic_socket_streambuf, 548
basic_stream_socket, 608
socket_base, 1082
default_buffer_size
buffered_read_stream, 709
buffered_write_stream, 727
default_workarounds

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ssl::context, 1093
ssl::context_base, 1108
destroy
datagram_socket_service, 762
deadline_timer_service, 772
ip::resolver_service, 883
posix::stream_descriptor_service, 999
raw_socket_service, 1008
seq_packet_socket_service, 1047
serial_port_service, 1064
signal_set_service, 1071
socket_acceptor_service, 1075
stream_socket_service, 1139
waitable_timer_service, 1156
windows::object_handle_service, 1218
windows::random_access_handle_service, 1228
windows::stream_handle_service, 1234
difference_type
buffers_iterator, 733
ip::basic_resolver_iterator, 854
dispatch
io_service, 789
io_service::strand, 801
do_not_route
basic_datagram_socket, 228
basic_raw_socket, 313
basic_seq_packet_socket, 375
basic_socket, 459
basic_socket_acceptor, 508
basic_socket_streambuf, 549
basic_stream_socket, 608
socket_base, 1082
duration
basic_waitable_timer, 662
waitable_timer_service, 1156
duration_type
basic_deadline_timer, 277
basic_socket_iostream, 531
basic_socket_streambuf, 550
deadline_timer_service, 772
time_traits< boost::posix_time::ptime >, 1150

E

enable_connection_aborted
basic_datagram_socket, 229
basic_raw_socket, 314
basic_seq_packet_socket, 375
basic_socket, 460
basic_socket_acceptor, 509
basic_socket_streambuf, 550
basic_stream_socket, 609
socket_base, 1083
end
buffers_iterator, 733
const_buffers_1, 754
mutable_buffers_1, 947
null_buffers, 950

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

endpoint
ip::basic_resolver_entry, 851
ip::icmp, 866
ip::tcp, 893
ip::udp, 904
local::datagram_protocol, 924
local::stream_protocol, 937
endpoint_type
basic_datagram_socket, 229
basic_raw_socket, 314
basic_seq_packet_socket, 376
basic_socket, 461
basic_socket_acceptor, 509
basic_socket_iostream, 531
basic_socket_streambuf, 550
basic_stream_socket, 609
datagram_socket_service, 762
ip::basic_resolver, 843
ip::basic_resolver_entry, 851
ip::resolver_service, 883
raw_socket_service, 1009
seq_packet_socket_service, 1047
socket_acceptor_service, 1075
stream_socket_service, 1139
eof
error::misc_errors, 779
error
basic_socket_iostream, 531
basic_socket_streambuf, 551
error::addrinfo_category, 775
error::addrinfo_errors, 775
error::basic_errors, 775
error::get_addrinfo_category, 776
error::get_misc_category, 777
error::get_netdb_category, 777
error::get_ssl_category, 777
error::get_system_category, 777
error::make_error_code, 778
error::misc_category, 779
error::misc_errors, 779
error::netdb_category, 779
error::netdb_errors, 779
error::ssl_category, 780
error::ssl_errors, 780
error::system_category, 780
even
serial_port_base::parity, 1060
expires_at
basic_deadline_timer, 277
basic_socket_iostream, 532
basic_socket_streambuf, 551
basic_waitable_timer, 662
deadline_timer_service, 772
waitable_timer_service, 1156
expires_from_now
basic_deadline_timer, 279
basic_socket_iostream, 532
basic_socket_streambuf, 551

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_waitable_timer, 663
deadline_timer_service, 773
waitable_timer_service, 1157

F

family
ip::icmp, 868
ip::tcp, 894
ip::udp, 906
local::datagram_protocol, 926
local::stream_protocol, 938
fault
error::basic_errors, 775
fd_set_failure
error::misc_errors, 779
file_format
ssl::context, 1093
ssl::context_base, 1108
fill
buffered_read_stream, 709
buffered_stream, 718
flags
ip::basic_resolver_query, 863
ip::resolver_query_base, 879
flow_control
serial_port_base::flow_control, 1058
flush
buffered_stream, 718
buffered_write_stream, 727
fork_child
io_service, 789
fork_event
io_service, 789
fork_parent
io_service, 789
fork_prepare
io_service, 789
fork_service
io_service::service, 800
for_reading
ssl::context, 1096
ssl::context_base, 1110
for_writing
ssl::context, 1096
ssl::context_base, 1110
from_string
ip::address, 808
ip::address_v4, 817
ip::address_v6, 825

G

get
windows::overlapped_ptr, 1221
get_implementation
basic_datagram_socket, 230
basic_deadline_timer, 280
basic_io_object, 285

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket, 315
basic_seq_packet_socket, 376
basic_serial_port, 420
basic_signal_set, 440
basic_socket, 461
basic_socket_acceptor, 509
basic_socket_streambuf, 552
basic_stream_socket, 610
basic_waitable_timer, 665
ip::basic_resolver, 843
posix::basic_descriptor, 959
posix::basic_stream_descriptor, 978
windows::basic_handle, 1164
windows::basic_object_handle, 1176
windows::basic_random_access_handle, 1189
windows::basic_stream_handle, 1205
get_io_service
basic_datagram_socket, 230
basic_deadline_timer, 281
basic_io_object, 286
basic_raw_socket, 315
basic_seq_packet_socket, 377
basic_serial_port, 420
basic_signal_set, 440
basic_socket, 461
basic_socket_acceptor, 510
basic_socket_streambuf, 553
basic_stream_socket, 610
basic_waitable_timer, 665
buffered_read_stream, 709
buffered_stream, 719
buffered_write_stream, 727
datagram_socket_service, 762
deadline_timer_service, 774
io_service::service, 799
io_service::strand, 802
io_service::work, 804
ip::basic_resolver, 843
ip::resolver_service, 883
posix::basic_descriptor, 959
posix::basic_stream_descriptor, 978
posix::stream_descriptor_service, 999
raw_socket_service, 1009
seq_packet_socket_service, 1047
serial_port_service, 1064
signal_set_service, 1071
socket_acceptor_service, 1075
ssl::stream, 1116
stream_socket_service, 1139
waitable_timer_service, 1158
windows::basic_handle, 1164
windows::basic_object_handle, 1176
windows::basic_random_access_handle, 1190
windows::basic_stream_handle, 1206
windows::object_handle_service, 1218
windows::random_access_handle_service, 1228
windows::stream_handle_service, 1235
get_option

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_datagram_socket, 230
basic_raw_socket, 315
basic_seq_packet_socket, 377
basic_serial_port, 420
basic_socket, 461
basic_socket_acceptor, 510
basic_socket_streambuf, 553
basic_stream_socket, 610
datagram_socket_service, 762
raw_socket_service, 1009
seq_packet_socket_service, 1047
serial_port_service, 1064
socket_acceptor_service, 1075
stream_socket_service, 1139
get_service
basic_datagram_socket, 232
basic_deadline_timer, 281
basic_io_object, 286
basic_raw_socket, 317
basic_seq_packet_socket, 378
basic_serial_port, 421
basic_signal_set, 441
basic_socket, 463
basic_socket_acceptor, 511
basic_socket_streambuf, 554
basic_stream_socket, 612
basic_waitable_timer, 666
ip::basic_resolver, 843
posix::basic_descriptor, 959
posix::basic_stream_descriptor, 978
windows::basic_handle, 1165
windows::basic_object_handle, 1176
windows::basic_random_access_handle, 1190
windows::basic_stream_handle, 1206

H

handshake
ssl::stream, 1116
handshake_type
ssl::stream, 1117
ssl::stream_base, 1126
hardware
serial_port_base::flow_control, 1059
has_service, 780
io_service, 790
high_resolution_timer, 781
hints
ip::basic_resolver_query, 863
host_name
ip::basic_resolver_entry, 852
ip::basic_resolver_query, 863
host_not_found
error::netdb_errors, 779
host_not_found_try_again
error::netdb_errors, 779
host_unreachable
error::basic_errors, 775

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

I

id
datagram_socket_service, 762
deadline_timer_service, 774
io_service::id, 798
ip::resolver_service, 883
posix::stream_descriptor_service, 999
raw_socket_service, 1009
seq_packet_socket_service, 1048
serial_port_service, 1064
signal_set_service, 1071
socket_acceptor_service, 1076
stream_socket_service, 1140
waitable_timer_service, 1158
windows::object_handle_service, 1218
windows::random_access_handle_service, 1228
windows::stream_handle_service, 1235
impl
ssl::context, 1093
ssl::stream, 1117
implementation
basic_datagram_socket, 232
basic_deadline_timer, 282
basic_io_object, 286
basic_raw_socket, 317
basic_seq_packet_socket, 379
basic_serial_port, 422
basic_signal_set, 441
basic_socket, 463
basic_socket_acceptor, 512
basic_socket_streambuf, 555
basic_stream_socket, 612
basic_waitable_timer, 666
ip::basic_resolver, 844
posix::basic_descriptor, 960
posix::basic_stream_descriptor, 979
windows::basic_handle, 1165
windows::basic_object_handle, 1177
windows::basic_random_access_handle, 1190
windows::basic_stream_handle, 1206
implementation_type
basic_datagram_socket, 232
basic_deadline_timer, 282
basic_io_object, 286
basic_raw_socket, 317
basic_seq_packet_socket, 379
basic_serial_port, 422
basic_signal_set, 441
basic_socket, 463
basic_socket_acceptor, 512
basic_socket_streambuf, 555
basic_stream_socket, 612
basic_waitable_timer, 666
datagram_socket_service, 762
deadline_timer_service, 774
ip::basic_resolver, 844
ip::resolver_service, 883

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_descriptor, 960
posix::basic_stream_descriptor, 979
posix::stream_descriptor_service, 999
raw_socket_service, 1009
seq_packet_socket_service, 1048
serial_port_service, 1064
signal_set_service, 1072
socket_acceptor_service, 1076
stream_socket_service, 1140
waitable_timer_service, 1158
windows::basic_handle, 1165
windows::basic_object_handle, 1177
windows::basic_random_access_handle, 1191
windows::basic_stream_handle, 1207
windows::object_handle_service, 1218
windows::random_access_handle_service, 1228
windows::stream_handle_service, 1235
impl_type
ssl::context, 1093
ssl::stream, 1117
interrupted
error::basic_errors, 775
invalid_argument
error::basic_errors, 775
invalid_service_owner
invalid_service_owner, 785
in_avail
buffered_read_stream, 709
buffered_stream, 719
buffered_write_stream, 727
in_progress
error::basic_errors, 775
iostream
ip::tcp, 894
local::stream_protocol, 938
io_control
basic_datagram_socket, 233
basic_raw_socket, 318
basic_seq_packet_socket, 379
basic_socket, 464
basic_socket_acceptor, 512
basic_socket_streambuf, 555
basic_stream_socket, 613
datagram_socket_service, 763
posix::basic_descriptor, 960
posix::basic_stream_descriptor, 979
posix::stream_descriptor_service, 999
raw_socket_service, 1009
seq_packet_socket_service, 1048
socket_acceptor_service, 1076
stream_socket_service, 1140
io_handler
basic_socket_streambuf, 556
io_service
io_service, 790
ip::host_name, 864
ip::multicast::enable_loopback, 875
ip::multicast::hops, 876

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::multicast::join_group, 876
ip::multicast::leave_group, 877
ip::multicast::outbound_interface, 877
ip::unicast::hops, 913
ip::v6_only, 914
is_class_a
  ip::address_v4, 818
is_class_b
  ip::address_v4, 818
is_class_c
  ip::address_v4, 818
is_link_local
  ip::address_v6, 826
is_loopback
  ip::address, 809
  ip::address_v4, 818
  ip::address_v6, 827
is_multicast
  ip::address, 809
  ip::address_v4, 819
  ip::address_v6, 827
is_multicast_global
  ip::address_v6, 827
is_multicast_link_local
  ip::address_v6, 827
is_multicast_node_local
  ip::address_v6, 827
is_multicast_org_local
  ip::address_v6, 827
is_multicast_site_local
  ip::address_v6, 827
is_open
  basic_datagram_socket, 234
  basic_raw_socket, 319
  basic_seq_packet_socket, 381
  basic_serial_port, 422
  basic_socket, 465
  basic_socket_acceptor, 514
  basic_socket_streambuf, 557
  basic_stream_socket, 614
  datagram_socket_service, 763
  posix::basic_descriptor, 962
  posix::basic_stream_descriptor, 981
  posix::stream_descriptor_service, 1000
  raw_socket_service, 1010
  seq_packet_socket_service, 1048
  serial_port_service, 1065
  socket_acceptor_service, 1076
  stream_socket_service, 1140
  windows::basic_handle, 1165
  windows::basic_object_handle, 1177
  windows::basic_random_access_handle, 1191
  windows::basic_stream_handle, 1207
  windows::object_handle_service, 1219
  windows::random_access_handle_service, 1228
  windows::stream_handle_service, 1235
is_site_local
  ip::address_v6, 827

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

is_unspecified
ip::address, 809
ip::address_v4, 819
ip::address_v6, 827
is_v4
ip::address, 809
is_v4_compatible
ip::address_v6, 828
is_v4_mapped
ip::address_v6, 828
is_v6
ip::address, 809
iterator
ip::basic_resolver, 844
iterator_category
buffers_iterator, 734
ip::basic_resolver_iterator, 855
iterator_type
ip::resolver_service, 883

K

keep_alive
basic_datagram_socket, 234
basic_raw_socket, 319
basic_seq_packet_socket, 381
basic_socket, 465
basic_socket_acceptor, 514
basic_socket_streambuf, 557
basic_stream_socket, 614
socket_base, 1084

L

less_than
time_traits< boost::posix_time::ptime >, 1150
linger
basic_datagram_socket, 235
basic_raw_socket, 320
basic_seq_packet_socket, 381
basic_socket, 466
basic_socket_acceptor, 515
basic_socket_streambuf, 557
basic_stream_socket, 615
socket_base, 1084
listen
basic_socket_acceptor, 515
socket_acceptor_service, 1076
load
serial_port_base::baud_rate, 1056
serial_port_base::character_size, 1057
serial_port_base::flow_control, 1058
serial_port_base::parity, 1059
serial_port_base::stop_bits, 1061
load_verify_file
ssl::context, 1094
local::connect_pair, 923
local_endpoint
basic_datagram_socket, 235

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket, 320
basic_seq_packet_socket, 382
basic_socket, 466
basic_socket_acceptor, 516
basic_socket_streambuf, 558
basic_stream_socket, 615
datagram_socket_service, 763
raw_socket_service, 1010
seq_packet_socket_service, 1048
socket_acceptor_service, 1076
stream_socket_service, 1140
loopback
ip::address_v4, 819
ip::address_v6, 828
lowest_layer
basic_datagram_socket, 236
basic_raw_socket, 321
basic_seq_packet_socket, 383
basic_serial_port, 422
basic_socket, 467
basic_socket_streambuf, 559
basic_stream_socket, 616
buffered_read_stream, 710
buffered_stream, 719
buffered_write_stream, 728
posix::basic_descriptor, 962
posix::basic_stream_descriptor, 981
ssl::stream, 1117
windows::basic_handle, 1166
windows::basic_object_handle, 1177
windows::basic_random_access_handle, 1191
windows::basic_stream_handle, 1207
lowest_layer_type
basic_datagram_socket, 237
basic_raw_socket, 322
basic_seq_packet_socket, 384
basic_serial_port, 423
basic_socket, 468
basic_socket_streambuf, 560
basic_stream_socket, 617
buffered_read_stream, 710
buffered_stream, 720
buffered_write_stream, 728
posix::basic_descriptor, 962
posix::basic_stream_descriptor, 981
ssl::stream, 1118
windows::basic_handle, 1166
windows::basic_object_handle, 1178
windows::basic_random_access_handle, 1192
windows::basic_stream_handle, 1208

M

max_connections
basic_datagram_socket, 242
basic_raw_socket, 327
basic_seq_packet_socket, 389
basic_socket, 473

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_acceptor, 517
basic_socket_streambuf, 565
basic_stream_socket, 622
socket_base, 1085
max_size
basic_streambuf, 652
message_do_not_route
basic_datagram_socket, 242
basic_raw_socket, 327
basic_seq_packet_socket, 389
basic_socket, 473
basic_socket_acceptor, 517
basic_socket_streambuf, 565
basic_stream_socket, 622
socket_base, 1085
message_end_of_record
basic_datagram_socket, 242
basic_raw_socket, 327
basic_seq_packet_socket, 389
basic_socket, 473
basic_socket_acceptor, 517
basic_socket_streambuf, 565
basic_stream_socket, 622
socket_base, 1085
message_flags
basic_datagram_socket, 242
basic_raw_socket, 327
basic_seq_packet_socket, 389
basic_socket, 473
basic_socket_acceptor, 518
basic_socket_streambuf, 565
basic_stream_socket, 622
socket_base, 1085
message_out_of_band
basic_datagram_socket, 242
basic_raw_socket, 327
basic_seq_packet_socket, 389
basic_socket, 473
basic_socket_acceptor, 518
basic_socket_streambuf, 565
basic_stream_socket, 622
socket_base, 1085
message_peek
basic_datagram_socket, 242
basic_raw_socket, 327
basic_seq_packet_socket, 389
basic_socket, 473
basic_socket_acceptor, 518
basic_socket_streambuf, 565
basic_stream_socket, 622
socket_base, 1085
message_size
error::basic_errors, 775
method
ssl::context, 1094
ssl::context_base, 1109
move_assign
datagram_socket_service, 763

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::stream_descriptor_service, 1000
raw_socket_service, 1010
seq_packet_socket_service, 1048
serial_port_service, 1065
socket_acceptor_service, 1077
stream_socket_service, 1140
windows::object_handle_service, 1219
windows::random_access_handle_service, 1228
windows::stream_handle_service, 1235
move_construct
datagram_socket_service, 763
posix::stream_descriptor_service, 1000
raw_socket_service, 1010
seq_packet_socket_service, 1049
serial_port_service, 1065
socket_acceptor_service, 1077
stream_socket_service, 1141
windows::object_handle_service, 1219
windows::random_access_handle_service, 1228
windows::stream_handle_service, 1235
mutable_buffer
mutable_buffer, 945
mutable_buffers_1
mutable_buffers_1, 947
mutable_buffers_type
basic_streambuf, 653

N

name_too_long
error::basic_errors, 775
native
basic_datagram_socket, 243
basic_raw_socket, 328
basic_seq_packet_socket, 390
basic_serial_port, 425
basic_socket, 474
basic_socket_acceptor, 518
basic_socket_streambuf, 566
basic_stream_socket, 623
datagram_socket_service, 763
posix::basic_descriptor, 965
posix::basic_stream_descriptor, 984
posix::stream_descriptor_service, 1000
raw_socket_service, 1010
seq_packet_socket_service, 1049
serial_port_service, 1065
socket_acceptor_service, 1077
stream_socket_service, 1141
windows::basic_handle, 1168
windows::basic_object_handle, 1180
windows::basic_random_access_handle, 1194
windows::basic_stream_handle, 1210
windows::random_access_handle_service, 1229
windows::stream_handle_service, 1236
native_handle
basic_datagram_socket, 243
basic_raw_socket, 328

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_seq_packet_socket, 390
basic_serial_port, 425
basic_socket, 474
basic_socket_acceptor, 518
basic_socket_streambuf, 566
basic_stream_socket, 623
datagram_socket_service, 764
posix::basic_descriptor, 965
posix::basic_stream_descriptor, 984
posix::stream_descriptor_service, 1000
raw_socket_service, 1010
seq_packet_socket_service, 1049
serial_port_service, 1065
socket_acceptor_service, 1077
ssl::context, 1095
ssl::stream, 1118
ssl::verify_context, 1127
stream_socket_service, 1141
windows::basic_handle, 1168
windows::basic_object_handle, 1180
windows::basic_random_access_handle, 1194
windows::basic_stream_handle, 1210
windows::object_handle_service, 1219
windows::random_access_handle_service, 1229
windows::stream_handle_service, 1236
native_handle_type
basic_datagram_socket, 243
basic_raw_socket, 328
basic_seq_packet_socket, 390
basic_serial_port, 425
basic_socket, 474
basic_socket_acceptor, 518
basic_socket_streambuf, 566
basic_stream_socket, 623
datagram_socket_service, 764
posix::basic_descriptor, 965
posix::basic_stream_descriptor, 984
posix::stream_descriptor_service, 1000
raw_socket_service, 1011
seq_packet_socket_service, 1049
serial_port_service, 1065
socket_acceptor_service, 1077
ssl::context, 1095
ssl::stream, 1118
ssl::verify_context, 1128
stream_socket_service, 1141
windows::basic_handle, 1168
windows::basic_object_handle, 1180
windows::basic_random_access_handle, 1194
windows::basic_stream_handle, 1210
windows::object_handle_service, 1219
windows::random_access_handle_service, 1229
windows::stream_handle_service, 1236
native_non_blocking
basic_datagram_socket, 243
basic_raw_socket, 328
basic_seq_packet_socket, 390
basic_socket, 474

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_socket_acceptor, 519
basic_socket_streambuf, 566
basic_stream_socket, 623
datagram_socket_service, 764
posix::basic_descriptor, 966
posix::basic_stream_descriptor, 985
posix::stream_descriptor_service, 1001
raw_socket_service, 1011
seq_packet_socket_service, 1049
socket_acceptor_service, 1077
stream_socket_service, 1141
native_type
basic_datagram_socket, 250
basic_raw_socket, 335
basic_seq_packet_socket, 397
basic_serial_port, 426
basic_socket, 480
basic_socket_acceptor, 520
basic_socket_streambuf, 573
basic_stream_socket, 630
datagram_socket_service, 765
posix::basic_descriptor, 967
posix::basic_stream_descriptor, 986
posix::stream_descriptor_service, 1001
raw_socket_service, 1011
seq_packet_socket_service, 1050
serial_port_service, 1066
socket_acceptor_service, 1078
stream_socket_service, 1142
windows::basic_handle, 1169
windows::basic_object_handle, 1180
windows::basic_random_access_handle, 1194
windows::basic_stream_handle, 1210
windows::random_access_handle_service, 1229
windows::stream_handle_service, 1236
netmask
ip::address_v4, 819
network_down
error::basic_errors, 775
network_reset
error::basic_errors, 775
network_unreachable
error::basic_errors, 775
next_layer
buffered_read_stream, 710
buffered_stream, 720
buffered_write_stream, 728
ssl::stream, 1119
next_layer_type
buffered_read_stream, 711
buffered_stream, 720
buffered_write_stream, 729
ssl::stream, 1119
none
serial_port_base::flow_control, 1059
serial_port_base::parity, 1060
non_blocking
basic_datagram_socket, 250

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_raw_socket, 335
basic_seq_packet_socket, 397
basic_socket, 480
basic_socket_acceptor, 520
basic_socket_streambuf, 573
basic_stream_socket, 630
datagram_socket_service, 765
posix::basic_descriptor, 967
posix::basic_stream_descriptor, 986
posix::stream_descriptor_service, 1001
raw_socket_service, 1012
seq_packet_socket_service, 1050
socket_acceptor_service, 1078
stream_socket_service, 1142
non_blocking_io
basic_datagram_socket, 251
basic_raw_socket, 336
basic_seq_packet_socket, 398
basic_socket, 481
basic_socket_acceptor, 522
basic_socket_streambuf, 574
basic_stream_socket, 631
posix::basic_descriptor, 968
posix::basic_stream_descriptor, 988
posix::descriptor_base, 993
socket_base, 1085
notify_fork
io_service, 791
not_connected
error::basic_errors, 775
not_found
error::misc_errors, 779
not_socket
error::basic_errors, 775
now
time_traits< boost::posix_time::ptime >, 1150
no_buffer_space
error::basic_errors, 775
no_data
error::netdb_errors, 779
no_delay
ip::tcp, 895
no_descriptors
error::basic_errors, 775
no_memory
error::basic_errors, 775
no_permission
error::basic_errors, 775
no_protocol_option
error::basic_errors, 775
no_recovery
error::netdb_errors, 779
no_sslv2
ssl::context, 1095
ssl::context_base, 1109
no_sslv3
ssl::context, 1096
ssl::context_base, 1109

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

no_tlsv1
ssl::context, 1096
ssl::context_base, 1109
numeric_host
ip::basic_resolver_query, 863
ip::resolver_query_base, 880
numeric_service
ip::basic_resolver_query, 863
ip::resolver_query_base, 880

O

object_handle_service
windows::object_handle_service, 1219
odd
serial_port_base::parity, 1060
one
serial_port_base::stop_bits, 1061
onepointfive
serial_port_base::stop_bits, 1061
open
basic_datagram_socket, 252
basic_raw_socket, 337
basic_seq_packet_socket, 399
basic_serial_port, 426
basic_socket, 482
basic_socket_acceptor, 522
basic_socket_streambuf, 575
basic_stream_socket, 632
datagram_socket_service, 765
raw_socket_service, 1012
seq_packet_socket_service, 1051
serial_port_service, 1066
socket_acceptor_service, 1079
stream_socket_service, 1143
operation_aborted
error::basic_errors, 775
operation_not_supported
error::basic_errors, 775
operator *
buffers_iterator, 734
ip::basic_resolver_iterator, 855
operator endpoint_type
ip::basic_resolver_entry, 852
operator!=
buffers_iterator, 734
ip::address, 809
ip::address_v4, 819
ip::address_v6, 828
ip::basic_endpoint, 836
ip::basic_resolver_iterator, 855
ip::icmp, 868
ip::tcp, 896
ip::udp, 906
local::basic_endpoint, 919
operator()
ssl::rfc2818_verification, 1111
operator+

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

buffers_iterator, 734
const_buffer, 751
const_buffers_1, 754
mutable_buffer, 945
mutable_buffers_1, 948
operator++
buffers_iterator, 735
ip::basic_resolver_iterator, 855
operator+=
buffers_iterator, 735
operator-
buffers_iterator, 736
operator--
buffers_iterator, 736
operator-=
buffers_iterator, 737
operator->
buffers_iterator, 737
ip::basic_resolver_iterator, 856
operator<
buffers_iterator, 737
ip::address, 810
ip::address_v4, 819
ip::address_v6, 828
ip::basic_endpoint, 836
local::basic_endpoint, 920
operator<<
ip::address, 810
ip::address_v4, 820
ip::address_v6, 828
ip::basic_endpoint, 836
local::basic_endpoint, 920
operator<=
buffers_iterator, 737
ip::address, 810
ip::address_v4, 820
ip::address_v6, 829
ip::basic_endpoint, 837
local::basic_endpoint, 920
operator=
basic_datagram_socket, 253
basic_io_object, 287
basic_raw_socket, 338
basic_seq_packet_socket, 400
basic_serial_port, 427
basic_socket, 483
basic_socket_acceptor, 523
basic_stream_socket, 633
ip::address, 811
ip::address_v4, 820
ip::address_v6, 829
ip::basic_endpoint, 837
local::basic_endpoint, 920
posix::basic_descriptor, 969
posix::basic_stream_descriptor, 988
ssl::context, 1096
windows::basic_handle, 1169
windows::basic_object_handle, 1181

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

windows::basic_random_access_handle, 1194
windows::basic_stream_handle, 1210
operator==
buffers_iterator, 738
ip::address, 811
ip::address_v4, 820
ip::address_v6, 829
ip::basic_endpoint, 837
ip::basic_resolver_iterator, 856
ip::icmp, 868
ip::tcp, 896
ip::udp, 907
local::basic_endpoint, 921
operator>
buffers_iterator, 738
ip::address, 812
ip::address_v4, 821
ip::address_v6, 829
ip::basic_endpoint, 837
local::basic_endpoint, 921
operator>=
buffers_iterator, 738
ip::address, 812
ip::address_v4, 821
ip::address_v6, 830
ip::basic_endpoint, 837
local::basic_endpoint, 921
operator[]
buffers_iterator, 738
options
ssl::context, 1096
ssl::context_base, 1109
overflow
basic_socket_streambuf, 576
basic_streambuf, 653
overlapped_ptr
windows::overlapped_ptr, 1221

P

parity
serial_port_base::parity, 1060
passive
ip::basic_resolver_query, 864
ip::resolver_query_base, 880
password_purpose
ssl::context, 1096
ssl::context_base, 1110
path
local::basic_endpoint, 921
peek
buffered_read_stream, 711
buffered_stream, 720
buffered_write_stream, 729
pem
ssl::context, 1093
ssl::context_base, 1108
placeholders::bytes_transferred, 951

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

placeholders::error, 951
placeholders::iterator, 952
placeholders::signal_number, 952
pointer
  buffers_iterator, 739
  ip::basic_resolver_iterator, 856
poll
  io_service, 791
poll_one
  io_service, 792
port
  ip::basic_endpoint, 838
posix::stream_descriptor, 994
post
  io_service, 793
  io_service::strand, 802
prepare
  basic_streambuf, 653
protocol
  ip::basic_endpoint, 838
  ip::icmp, 868
  ip::tcp, 896
  ip::udp, 907
  local::basic_endpoint, 922
  local::datagram_protocol, 926
  local::stream_protocol, 939
protocol_type
  basic_datagram_socket, 253
  basic_raw_socket, 338
  basic_seq_packet_socket, 400
  basic_socket, 483
  basic_socket_acceptor, 523
  basic_socket_streambuf, 576
  basic_stream_socket, 633
  datagram_socket_service, 765
  ip::basic_endpoint, 838
  ip::basic_resolver, 846
  ip::basic_resolver_entry, 852
  ip::basic_resolver_query, 864
  ip::resolver_service, 885
  local::basic_endpoint, 922
  raw_socket_service, 1012
  seq_packet_socket_service, 1051
  socket_acceptor_service, 1079
  stream_socket_service, 1143
puberror
  basic_socket_streambuf, 576

Q

query
  ip::basic_resolver, 846
query_type
  ip::resolver_service, 885

R

random_access_handle_service
  windows::random_access_handle_service, 1229

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

raw_socket_service
  raw_socket_service, 1012
rdbuf
  basic_socket_iostream, 533
read, 1014
read_at, 1023
read_some
  basic_serial_port, 427
  basic_stream_socket, 633
  buffered_read_stream, 711
  buffered_stream, 721
  buffered_write_stream, 729
  posix::basic_stream_descriptor, 988
  posix::stream_descriptor_service, 1002
  serial_port_service, 1066
  ssl::stream, 1119
  windows::basic_stream_handle, 1211
  windows::stream_handle_service, 1236
read_some_at
  windows::basic_random_access_handle, 1195
  windows::random_access_handle_service, 1229
read_until, 1031
receive
  basic_datagram_socket, 253
  basic_raw_socket, 338
  basic_seq_packet_socket, 400
  basic_stream_socket, 635
  datagram_socket_service, 766
  raw_socket_service, 1013
  seq_packet_socket_service, 1051
  stream_socket_service, 1143
receive_buffer_size
  basic_datagram_socket, 256
  basic_raw_socket, 341
  basic_seq_packet_socket, 403
  basic_socket, 483
  basic_socket_acceptor, 524
  basic_socket_streambuf, 576
  basic_stream_socket, 637
  socket_base, 1086
receive_from
  basic_datagram_socket, 256
  basic_raw_socket, 341
  datagram_socket_service, 766
  raw_socket_service, 1013
receive_low_watermark
  basic_datagram_socket, 259
  basic_raw_socket, 344
  basic_seq_packet_socket, 404
  basic_socket, 484
  basic_socket_acceptor, 524
  basic_socket_streambuf, 577
  basic_stream_socket, 638
  socket_base, 1086
reference
  buffers_iterator, 739
  ip::basic_resolver_iterator, 856
release

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_descriptor, 969
posix::basic_stream_descriptor, 990
posix::stream_descriptor_service, 1002
windows::overlapped_ptr, 1222
remote_endpoint
basic_datagram_socket, 259
basic_raw_socket, 344
basic_seq_packet_socket, 404
basic_socket, 484
basic_socket_streambuf, 578
basic_stream_socket, 639
datagram_socket_service, 766
raw_socket_service, 1013
seq_packet_socket_service, 1051
stream_socket_service, 1143
remove
basic_signal_set, 441
signal_set_service, 1072
reserve
basic_streambuf, 653
reset
io_service, 793
windows::overlapped_ptr, 1222
resize
ip::basic_endpoint, 838
local::basic_endpoint, 922
resolve
ip::basic_resolver, 847
ip::resolver_service, 886
resolver
ip::icmp, 869
ip::tcp, 896
ip::udp, 907
resolver_service
ip::resolver_service, 887
result_type
ssl::rfc2818_verification, 1111
reuse_address
basic_datagram_socket, 260
basic_raw_socket, 345
basic_seq_packet_socket, 405
basic_socket, 486
basic_socket_acceptor, 525
basic_socket_streambuf, 579
basic_stream_socket, 640
socket_base, 1087
rfc2818_verification
ssl::rfc2818_verification, 1111
run
io_service, 794
run_one
io_service, 795

S

scope_id
ip::address_v6, 830
send

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

basic_datagram_socket, 261
basic_raw_socket, 346
basic_seq_packet_socket, 406
basic_stream_socket, 640
datagram_socket_service, 766
raw_socket_service, 1013
seq_packet_socket_service, 1051
stream_socket_service, 1143
send_break
basic_serial_port, 428
serial_port_service, 1066
send_buffer_size
basic_datagram_socket, 263
basic_raw_socket, 348
basic_seq_packet_socket, 407
basic_socket, 486
basic_socket_acceptor, 525
basic_socket_streambuf, 579
basic_stream_socket, 643
socket_base, 1087
send_low_watermark
basic_datagram_socket, 264
basic_raw_socket, 348
basic_seq_packet_socket, 408
basic_socket, 487
basic_socket_acceptor, 526
basic_socket_streambuf, 580
basic_stream_socket, 643
socket_base, 1088
send_to
basic_datagram_socket, 264
basic_raw_socket, 349
datagram_socket_service, 766
raw_socket_service, 1013
seq_packet_socket_service
seq_packet_socket_service, 1052
serial_port, 1052
serial_port_service
serial_port_service, 1066
server
ssl::stream, 1117
ssl::stream_base, 1126
service
basic_datagram_socket, 267
basic_deadline_timer, 282
basic_io_object, 287
basic_raw_socket, 351
basic_seq_packet_socket, 409
basic_serial_port, 429
basic_signal_set, 442
basic_socket, 487
basic_socket_acceptor, 527
basic_socket_streambuf, 580
basic_stream_socket, 644
basic_waitable_timer, 667
io_service::service, 799
ip::basic_resolver, 849
posix::basic_descriptor, 969

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_stream_descriptor, 990
windows::basic_handle, 1169
windows::basic_object_handle, 1181
windows::basic_random_access_handle, 1196
windows::basic_stream_handle, 1212
service_already_exists
service_already_exists, 1067
service_name
ip::basic_resolver_entry, 852
ip::basic_resolver_query, 864
service_not_found
error::addrinfo_errors, 775
service_type
basic_datagram_socket, 267
basic_deadline_timer, 282
basic_io_object, 287
basic_raw_socket, 351
basic_seq_packet_socket, 409
basic_serial_port, 429
basic_signal_set, 443
basic_socket, 488
basic_socket_acceptor, 527
basic_socket_streambuf, 581
basic_stream_socket, 644
basic_waitable_timer, 667
ip::basic_resolver, 850
posix::basic_descriptor, 970
posix::basic_stream_descriptor, 990
windows::basic_handle, 1169
windows::basic_object_handle, 1181
windows::basic_random_access_handle, 1197
windows::basic_stream_handle, 1213
setbuf
basic_socket_streambuf, 582
set_default_verify_paths
ssl::context, 1097
set_option
basic_datagram_socket, 267
basic_raw_socket, 352
basic_seq_packet_socket, 409
basic_serial_port, 429
basic_socket, 488
basic_socket_acceptor, 527
basic_socket_streambuf, 581
basic_stream_socket, 644
datagram_socket_service, 767
raw_socket_service, 1014
seq_packet_socket_service, 1052
serial_port_service, 1066
socket_acceptor_service, 1079
stream_socket_service, 1144
set_options
ssl::context, 1097
set_password_callback
ssl::context, 1098
set_verify_callback
ssl::context, 1100
ssl::stream, 1121

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

set_verify_mode
ssl::context, 1101
ssl::stream, 1122
shutdown
basic_datagram_socket, 268
basic_raw_socket, 353
basic_seq_packet_socket, 410
basic_socket, 489
basic_socket_streambuf, 582
basic_stream_socket, 646
datagram_socket_service, 767
raw_socket_service, 1014
seq_packet_socket_service, 1052
ssl::stream, 1123
stream_socket_service, 1144
shutdown_both
basic_datagram_socket, 270
basic_raw_socket, 354
basic_seq_packet_socket, 412
basic_socket, 490
basic_socket_acceptor, 529
basic_socket_streambuf, 584
basic_stream_socket, 647
socket_base, 1089
shutdown_receive
basic_datagram_socket, 270
basic_raw_socket, 354
basic_seq_packet_socket, 412
basic_socket, 490
basic_socket_acceptor, 529
basic_socket_streambuf, 584
basic_stream_socket, 647
socket_base, 1089
shutdown_send
basic_datagram_socket, 270
basic_raw_socket, 354
basic_seq_packet_socket, 412
basic_socket, 490
basic_socket_acceptor, 529
basic_socket_streambuf, 584
basic_stream_socket, 647
socket_base, 1089
shutdown_service
io_service::service, 800
shutdown_type
basic_datagram_socket, 270
basic_raw_socket, 354
basic_seq_packet_socket, 412
basic_socket, 490
basic_socket_acceptor, 528
basic_socket_streambuf, 584
basic_stream_socket, 647
socket_base, 1089
shut_down
error::basic_errors, 775
signal_set, 1067
signal_set_service
signal_set_service, 1072

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

single_dh_use
ssl::context, 1102
ssl::context_base, 1110
size
basic_streambuf, 653
ip::basic_endpoint, 839
local::basic_endpoint, 923
socket
ip::icmp, 870
ip::tcp, 898
ip::udp, 908
local::datagram_protocol, 926
local::stream_protocol, 939
socket_acceptor_service
socket_acceptor_service, 1079
socket_type_not_supported
error::addrinfo_errors, 775
software
serial_port_base::flow_control, 1059
ssl
ssl::stream::impl_struct, 1126
ssl::verify_client_once, 1127
ssl::verify_fail_if_no_peer_cert, 1128
ssl::verify_mode, 1128
ssl::verify_none, 1129
ssl::verify_peer, 1129
sslv2
ssl::context, 1095
ssl::context_base, 1109
sslv23
ssl::context, 1095
ssl::context_base, 1109
sslv23_client
ssl::context, 1095
ssl::context_base, 1109
sslv23_server
ssl::context, 1095
ssl::context_base, 1109
sslv2_client
ssl::context, 1095
ssl::context_base, 1109
sslv2_server
ssl::context, 1095
ssl::context_base, 1109
sslv3
ssl::context, 1095
ssl::context_base, 1109
sslv3_client
ssl::context, 1095
ssl::context_base, 1109
sslv3_server
ssl::context, 1095
ssl::context_base, 1109
steady_timer, 1129
stop
io_service, 796
stopped
io_service, 796

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

stop_bits
  serial_port_base::stop_bits, 1061
store
  serial_port_base::baud_rate, 1056
  serial_port_base::character_size, 1057
  serial_port_base::flow_control, 1058
  serial_port_base::parity, 1060
  serial_port_base::stop_bits, 1061
strand, 1132
  io_service::strand, 802
stream
  ssl::stream, 1124
streambuf, 1144
stream_descriptor_service
  posix::stream_descriptor_service, 1002
stream_handle_service
  windows::stream_handle_service, 1236
stream_socket_service
  stream_socket_service, 1144
subtract
  time_traits< boost::posix_time::ptime >, 1151
sync
  basic_socket_streambuf, 584
system_timer, 1146

T

timed_out
  error::basic_errors, 775
timer_handler
  basic_socket_streambuf, 584
time_point
  basic_waitable_timer, 667
  waitable_timer_service, 1158
time_type
  basic_deadline_timer, 282
  basic_socket_iostream, 533
  basic_socket_streambuf, 584
  deadline_timer_service, 774
  time_traits< boost::posix_time::ptime >, 1151
tlsv1
  ssl::context, 1095
  ssl::context_base, 1109
tlsv1_client
  ssl::context, 1095
  ssl::context_base, 1109
tlsv1_server
  ssl::context, 1095
  ssl::context_base, 1109
to_bytes
  ip::address_v4, 821
  ip::address_v6, 830
to_posix_duration
  time_traits< boost::posix_time::ptime >, 1151
to_string
  ip::address, 812
  ip::address_v4, 821
  ip::address_v6, 831

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

to_ulong
  ip::address_v4, 822
to_v4
  ip::address, 812
  ip::address_v6, 831
to_v6
  ip::address, 813
to_wait_duration
  wait_traits, 1154
traits_type
  basic_deadline_timer, 283
  basic_waitable_timer, 667
  deadline_timer_service, 774
  waitable_timer_service, 1158
transfer_all, 1151
transfer_at_least, 1152
transfer_exactly, 1152
try_again
  error::basic_errors, 775
two
  serial_port_base::stop_bits, 1061
type
  ip::icmp, 875
  ip::tcp, 903
  ip::udp, 913
  local::datagram_protocol, 931
  local::stream_protocol, 944
  serial_port_base::flow_control, 1058
  serial_port_base::parity, 1060
  serial_port_base::stop_bits, 1061

U

underflow
  basic_socket_streambuf, 584
  basic_streambuf, 654
use_certificate_chain_file
  ssl::context, 1102
use_certificate_file
  ssl::context, 1103
use_private_key_file
  ssl::context, 1104
use_rsa_private_key_file
  ssl::context, 1105
use_service, 1153
  io_service, 796
use_tmp_dh_file
  ssl::context, 1106

V

v4
  ip::icmp, 875
  ip::tcp, 903
  ip::udp, 913
v4_compatible
  ip::address_v6, 831
v4_mapped
  ip::address_v6, 831

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

ip::basic_resolver_query, 864
ip::resolver_query_base, 880
v6
ip::icmp, 875
ip::tcp, 903
ip::udp, 913
value
boost::system::is_error_code_enum< boost::asio::error::addrinfo_errors >, 1255
boost::system::is_error_code_enum< boost::asio::error::basic_errors >, 1255
boost::system::is_error_code_enum< boost::asio::error::misc_errors >, 1256
boost::system::is_error_code_enum< boost::asio::error::netdb_errors >, 1256
boost::system::is_error_code_enum< boost::asio::error::ssl_errors >, 1257
is_match_condition, 915
is_read_buffered, 915
is_write_buffered, 916
serial_port_base::baud_rate, 1057
serial_port_base::character_size, 1057
serial_port_base::flow_control, 1059
serial_port_base::parity, 1060
serial_port_base::stop_bits, 1061
value_type
buffers_iterator, 739
const_buffers_1, 754
ip::basic_resolver_iterator, 857
mutable_buffers_1, 948
null_buffers, 950
verify_context
ssl::verify_context, 1128

W

wait
basic_deadline_timer, 283
basic_waitable_timer, 667
deadline_timer_service, 775
waitable_timer_service, 1159
windows::basic_object_handle, 1181
windows::object_handle_service, 1220
waitable_timer_service
waitable_timer_service, 1159
windows::object_handle, 1214
windows::random_access_handle, 1223
windows::stream_handle, 1230
work
io_service::work, 804
would_block
error::basic_errors, 775
wrap
io_service, 797
io_service::strand, 802
write, 1237
write_at, 1245
write_some
basic_serial_port, 430
basic_stream_socket, 647
buffered_read_stream, 712
buffered_stream, 722
buffered_write_stream, 730

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/

posix::basic_stream_descriptor, 991
posix::stream_descriptor_service, 1002
serial_port_service, 1067
ssl::stream, 1124
windows::basic_stream_handle, 1213
windows::stream_handle_service, 1237
write_some_at
windows::basic_random_access_handle, 1197
windows::random_access_handle_service, 1230

XML to PDF by RenderX XEP XSL-FO Formatter, visit us at http://www.renderx.com/