Wave

0.1

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

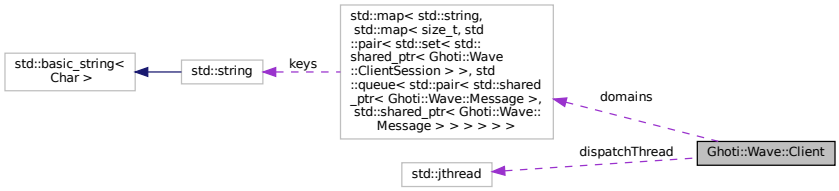
Class Documentation

3.1 Ghoti::Wave::Client Class Reference

Represents a client and all of its HTTP connections.

`#include <client.hpp>`

Collaboration diagram for Ghoti::Wave::Client:



Public Member Functions

- Client ()

  The constructor.
- ~Client ()

  The destructor.
- bool isRunning () const

  Indicates whether or not the client and its thread pools are currently active.
- Client & start ()

  Instructs the client to start its thread pool and begin processing the requests in its queue.
- Client & stop ()

  Instructs the client to gracefully shut down its thread pool.
- void dispatchLoop (std::stop_token stoken)

  The dispatch loop used by the thread pool to process sending requests and receiving responses.
- std::shared_ptr< Message > sendRequest (std::shared_ptr< Message > message)

  Enqueues a message to be sent to a client.

Private Attributes

- Ghoti::Pool::Pool workers

  The thread pool worker queue.
- std::map< std::string, std::map< size_t, std::pair< std::set< std::shared_ptr< Ghoti::Wave::ClientSession > >, std::queue< std::pair< std::shared_ptr< Message >, std::shared_ptr< Message > > > > > > domains

  Stores all connections and their request queues.
- std::jthread dispatchThread

  The thread that runs the read/write processing queues.
- bool running

  Whether or not the client is processing read/write actions from the sockets.

3.1.1 Detailed Description

Represents a client and all of its HTTP connections.

This class is currently only used for testing (so that the HTTP connection can be controlled explicitly), but that does not mean that it can't be used for more.

The Client object can establish connections to a server, receive message requests and forward them to the appropriate session, and report on the status of the connections.

This class exists primarily for testing the server, and as such offers fine-grained control of enabling and disabling features.

3.1.2 Member Function Documentation

3.1.2.1 dispatchLoop()

```
void Client::dispatchLoop (
            std::stop_token stoken )
```

The dispatch loop used by the thread pool to process sending requests and receiving responses.

Parameters

stoken	The jthread stop token, used to alert the thread that it should gracefully shut down.

3.1.2.2 isRunning()

```
bool Client::isRunning ( ) const
```

Indicates whether or not the client and its thread pools are currently active.

Returns

Whether or not the client and its thread pools are currently active.

3.1.2.3 sendRequest()

```
shared_ptr< Message > Client::sendRequest (
            std::shared_ptr< Message > message )
```

Enqueues a message to be sent to a client.

This returns a shared pointer to a Message which will contain the response when the request is completed.

Parameters

message	The request to be sent to a client.

Returns

A shared pointer to a Message the will eventually contain the response when the request is completed.

3.1.2.4 start()

```
Client& Ghoti::Wave::Client::start ( )
```

Instructs the client to start its thread pool and begin processing the requests in its queue.

Returns

The Client object.

3.1.2.5 stop()

```
Client & Client::stop ( )
```

Instructs the client to gracefully shut down its thread pool.

Returns

The Client object.

3.1.3 Member Data Documentation

3.1.3.1 domains

```
std::map<std::string, std::map<size_t, std::pair<std::set<std::shared_ptr<Ghoti::Wave::ClientSession>
>, std::queue<std::pair<std::shared_ptr<Message>, std::shared_ptr<Message> > > > > > > Ghoti↩
::Wave::Client::domains  [private]
```

Stores all connections and their request queues.

domains[domain][port] = {set{ClientSession}, queue{{request, response}}}

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

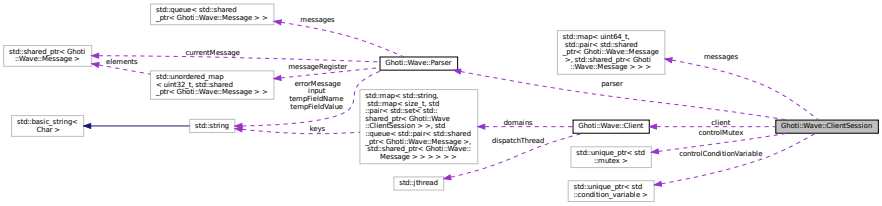
- include/wave/client.hpp
- src/client.cpp

3.2 Ghoti::Wave::ClientSession Class Reference

Represents a connection to a particular domain/port pair.

```
#include <clientSession.hpp>
```

Collaboration diagram for Ghoti::Wave::ClientSession:



Public Member Functions

- ClientSession (int hServer, Client *client)
  The constructor.
- ~ClientSession ()
  The destructor.
- bool hasReadDataWaiting ()
  Checks to see whether or not the session has data waiting to be read from the socket.
- bool hasWriteDataWaiting ()
  Checks to see whether or not the session has data waiting to be written to the socket.
- bool isFinished ()
  Indicates whether or not the session has completed all communications and may be terminated.
- void read ()
  Performs a read from the session.
- void write ()
  Performs a write to the session.
- void enqueue (std::shared_ptr< Message > request, std::shared_ptr< Message > response)
  Add a request/response pair to the session's queue.

Public Attributes

- std::unique_ptr< std::mutex > controlMutex

  Used to synchronize access to the session to make it thread safe.
- std::unique_ptr< std::condition_variable > controlConditionVariable

  Used to synchronize access to the session to make it thread safe.

Private Attributes

- int hServer

  The socket handle to the server.
- size_t requestSequence

  The index number of the next request to be enqueued.
- size_t writeSequence

  The index number of the current request being written.
- size_t writeOffset

  A byte offset, used to track how many bytes of a message have been written, so that individual write attempts do not duplicate data.
- size_t readSequence

  The index number of the current request being received.
- bool working

  Tracks whether or not the session has work queued.
- bool finished

  Tracks whether or not the session has completed all pending communications.
- Parser parser

  The parser object used to parse the raw HTTP stream.
- Client * client

  A pointer to the client object.
- std::map< uint64_t, std::pair< std::shared_ptr< Message >, std::shared_ptr< Message > > > messages

  Tracks message/response pairs.

3.2.1 Detailed Description

Represents a connection to a particular domain/port pair.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 ClientSession()

```
ClientSession::ClientSession (
            int hServer,
            Client * client )
```

The constructor.

The parent Client object will do the work of establishing the socket connection. Once the connection is established, then this class takes over the communication.

Parameters

hServer	The socket handle to the Server to which this session will communicate.
client	A pointer to the parent Client object.

3.2.3 Member Function Documentation

3.2.3.1 enqueue()

```
void ClientSession::enqueue (
            std::shared_ptr< Message > request,
            std::shared_ptr< Message > response )
```

Add a request/response pair to the session's queue.

The response object was created by the Client, and we will write our results into it as the request is processed.

Parameters

request	The HTTP request Message.
response	The HTTP response Message.

3.2.3.2 hasReadDataWaiting()

```
bool ClientSession::hasReadDataWaiting ( )
```

Checks to see whether or not the session has data waiting to be read from the socket.

This is non-blocking mutex controlled. If the session is currently working, then this function will return false.

Returns

Whether or not the session has data waiting to be read from the socket.

3.2.3.3 hasWriteDataWaiting()

```
bool ClientSession::hasWriteDataWaiting ( )
```

Checks to see whether or not the session has data waiting to be written to the socket.

This is non-blocking mutex controlled. If the session is currently working, then this function will return false.

Returns

Whether or not the session has data waiting to be written to the socket.

3.2.3.4 isFinished()

```
bool ClientSession::isFinished ( )
```

Indicates whether or not the session has completed all communications and may be terminated.

Returns

`true` if all communications have completed, `false` otherwise.

3.2.3.5 read()

```
void ClientSession::read ( )
```

Performs a read from the session.

This function is intended to be called by the client session's dispatch thread.

3.2.3.6 write()

```
void ClientSession::write ( )
```

Performs a write to the session.

This function is intended to be called by the client session's dispatch thread.

3.2.4 Member Data Documentation

3.2.4.1 messages

```
std::map<uint64_t, std::pair<std::shared_ptr<Message>, std::shared_ptr<Message> > > Ghoti↩
::Wave::ClientSession::messages  [private]
```

Tracks message/response pairs.

messages[request sequence #] = <request, response>

3.2.4.2 readSequence

```
size_t Ghoti::Wave::ClientSession::readSequence  [private]
```

The index number of the current request being received.

A session may send many requests before a single response is completely received. This variable tracks the reponse order so that it can be paired to the correct request.

3.2.4.3 requestSequence

```
size_t Ghoti::Wave::ClientSession::requestSequence  [private]
```

The index number of the next request to be enqueued.

A session may have multiple messages enqueued before the connection has been established. This variable ensures that messages are handled in the order requested.

3.2.4.4 writeSequence

```
size_t Ghoti::Wave::ClientSession::writeSequence  [private]
```

The index number of the current request being written.

A session must send requests in the order that they were enqueued. This variable tracks which message will be sent next.

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

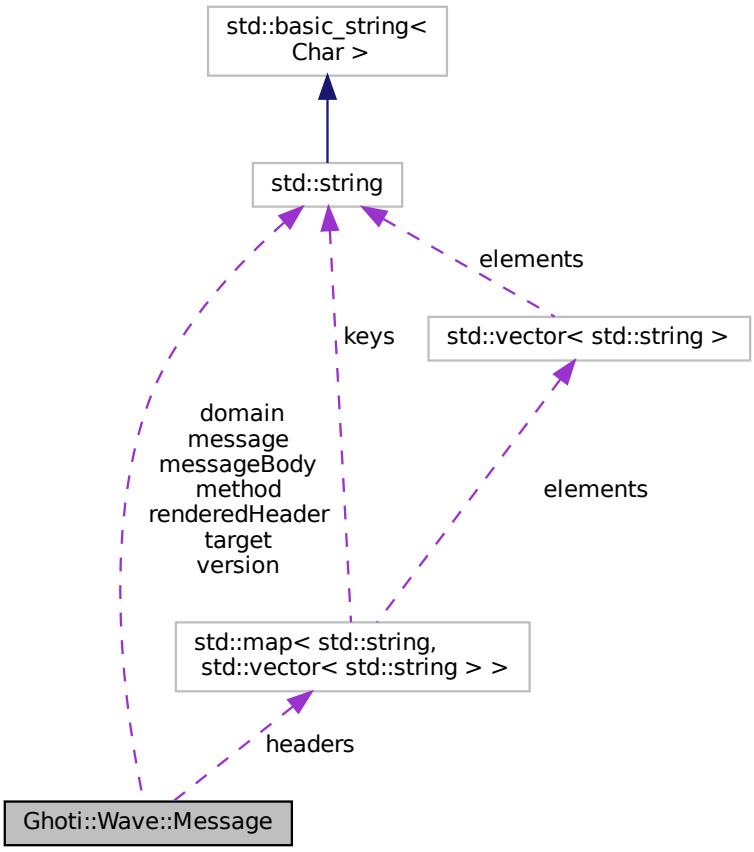
- include/wave/clientSession.hpp
- src/clientSession.cpp

3.3 Ghoti::Wave::Message Class Reference

Represents a HTTP message.

```
#include <message.hpp>
```

Collaboration diagram for Ghoti::Wave::Message:



Public Types

- enum Type { REQUEST , RESPONSE }

  Indicates whether the message is a request or a response.

Public Member Functions

- Message (Type type)

  The constructor.
- void adoptContents (Message &source)

  Move the contents of `source` into the `this` object, except for the promise and future attributes.
- const std::string & getRenderedHeader1 ()

  Get the HTTP/1.1 rendered header as a string.
- bool hasError () const

  Indicates that the message has an error.
- Message & setStatusCode (size_t statusCode)

  Set the status code of the message.
- size_t getStatusCode () const

Get the status code of the message.

- Message & setErrorMessage (const std::string &message)

  Set an error message description.

- Message & setMessage (const std::string &message)

  Set a status message.

- const std::string & getMessage () const

  Get the status message.

- Message & setMethod (const std::string &method)

  Set the HTTP method of the message.

- const std::string & getMethod () const

  Get the HTTP method of the message.

- Message & setTarget (const std::string &target)

  Set the URL target of the message.

- const std::string & getTarget () const

  Get the URL target of the message.

- Message & setVersion (const std::string &version)

  Set the HTTP version of the message.

- const std::string & getVersion () const

  Get the HTTP version of the message.

- void addFieldValue (const std::string &name, const std::string &value)

  Add a header key/value pair.

- const std::map< std::string, std::vector< std::string > > & getFields () const

  Get the map of all header field key/value pairs.

- Type getType () const

  Get the Message::Type of the message.

- Message & setMessageBody (const std::string &body)

  Set the content body of the message.

- const std::string & getMessageBody () const

  Get the content body of the message.

- size_t getContentLength () const

  Get the content length of the message body.

- Message & setPort (size_t port)

  Set the port to which the message is targeted.

- size_t getPort () const

  Get the port to which the message is targeted.

- Message & setDomain (const std::string &domain)

  Set the domain to which the message is targeted.

- const std::string & getDomain () const

  Get the domain to which the message is targeted.

- void setReady (bool isError)

  Notify the associated promise/future that the message is completed.

- std::future< bool > & getReadyFuture ()

  Get the future which will indicate when the message has been fully processed.

- Message & setId (uint32_t id)

  Set the ID of the message.

- uint32_t getId () const

  Get the ID of the message.

Private Attributes

- bool headerIsRendered

  Used to track whether or not the header has been rendered to a string.
- bool errorIsSet

  Tracks whether or not an error has been set.
- Type type

  The Message::Type of the message.
- uint32_t id

  The ID number of the message.
- size_t port

  The port to which the message is targeted.
- size_t statusCode

  The status code of the message.
- size_t contentLength

  The contentLength of the message.
- std::string renderedHeader

  A cached version of the HTTP/1.1 header.
- std::string message

  The status message.
- std::string method

  The HTTP method.
- std::string domain

  The domain target of the message.
- std::string target

  The URL target of the message.
- std::string version

  The HTTP version of the message.
- std::string messageBody

  The content body of the message.
- std::map< std::string, std::vector< std::string > > headers

  A collection of headers and their associated values.
- std::promise< bool > readyPromise

  The promise used for asynchronous notification of when the message has been processed.
- std::future< bool > readyFuture

  The future used for asynchronous notification of when the message has been processed.

3.3.1 Detailed Description

Represents a HTTP message.

3.3.2 Member Enumeration Documentation

3.3.2.1 Type

`enum Ghoti::Wave::Message::Type`

Indicates whether the message is a request or a response.

Enumerator

REQUEST	A HTTP Request.
RESPONSE	A HTTP Response.

3.3.3 Constructor & Destructor Documentation

3.3.3.1 Message()

```
Message::Message (
            Type type )
```

The constructor.

Messages must have an associated type.

Parameters

type	The Message::Type of the HTTP message.

3.3.4 Member Function Documentation

3.3.4.1 addFieldValue()

```
void Message::addFieldValue (
            const std::string & name,
            const std::string & value )
```

Add a header key/value pair.

Parameters

name	The field name.
value	The field value.

3.3.4.2 adoptContents()

```
void Message::adoptContents (
            Message & source )
```

Move the contents of `source` into the `this` object, except for the promise and future attributes.

This method is necessary because the parser may have already started populating a Message object. A client, however, must supply the Message object so that the client can know when the promise/future is fulfilled. The only way to accomplish this is to provide a way for the parser to have a provided Message "adopt" the contents of an existing message, but not bother the associated promise/future of the target.

Parameters

source	The Message whose contents will be adopted into `this`.

3.3.4.3 getContentLength()

```
size_t Message::getContentLength ( ) const
```

Get the content length of the message body.

Returns

The content length of the message body.

3.3.4.4 getDomain()

```
const string & Message::getDomain ( ) const
```

Get the domain to which the message is targeted.

Returns

The target domain.

3.3.4.5 getFields()

```
const map< string, vector< string > > & Message::getFields ( ) const
```

Get the map of all header field key/value pairs.

fields[field name] = [field value]

3.3.4.6 getId()

```
uint32_t Message::getId ( ) const
```

Get the ID of the message.

Returns

The ID number of the message.

3.3.4.7 getMessage()

```
const std::string & Message::getMessage ( ) const
```

Get the status message.

Returns

The status message.

3.3.4.8 getMessageBody()

```
const string & Message::getMessageBody ( ) const
```

Get the content body of the message.

Returns

The content body.

3.3.4.9 getMethod()

```
const std::string & Message::getMethod ( ) const
```

Get the HTTP method of the message.

Returns

The HTTP method.

3.3.4.10 getPort()

```
size_t Message::getPort ( ) const
```

Get the port to which the message is targeted.

Returns

The target port.

3.3.4.11 getReadyFuture()

```
future< bool > & Message::getReadyFuture ( )
```

Get the future which will indicate when the message has been fully processed.

Returns

The future used to monitor the status of the message.

3.3.4.12 getRenderedHeader1()

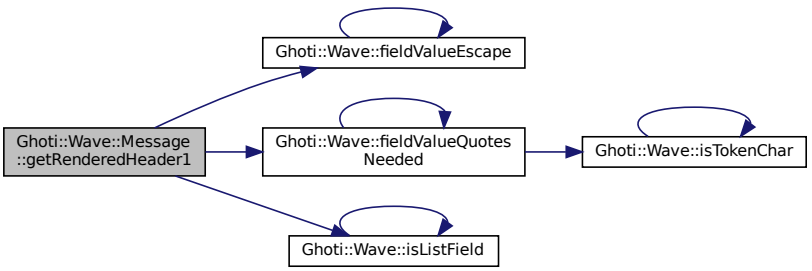
```
const string & Message::getRenderedHeader1 ( )
```

Get the HTTP/1.1 rendered header as a string.

Returns

A string containing the HTTP/1.1 rendered header.

Here is the call graph for this function:



3.3.4.13 getStatusCode()

```
size_t Message::getStatusCode ( ) const
```

Get the status code of the message.

Returns

The status code of the message.

3.3.4.14 getTarget()

```
const std::string & Message::getTarget ( ) const
```

Get the URL target of the message.

Returns

The URL target.

3.3.4.15 getType()

```
Message::Type Message::getType ( ) const
```

Get the Message::Type of the message.

Returns

The Message::Type of the message.

3.3.4.16 getVersion()

```
const std::string & Message::getVersion ( ) const
```

Get the HTTP version of the message.

Returns

The HTTP version.

3.3.4.17 hasError()

```
bool Message::hasError ( ) const
```

Indicates that the message has an error.

Returns

`true` if there is an error, `false` otherwise.

3.3.4.18 setDomain()

```
Message & Message::setDomain (
            const std::string & domain )
```

Set the domain to which the message is targeted.

Parameters

domain	The target domain.

Returns

The Message object.

3.3.4.19 setErrorMessage()

```
Message & Message::setErrorMessage (
            const std::string & message )
```

Set an error message description.

Parameters

message	The error message description.

Returns

The Message object.

3.3.4.20 setId()

```
Message & Message::setId (
            uint32_t id )
```

Set the ID of the message.

Parameters

id	The ID number of the message.

Returns

The Message object.

3.3.4.21 setMessage()

```
Message & Message::setMessage (
            const std::string & message )
```

Set a status message.

Parameters

The	status message description.

Returns

The Message object.

3.3.4.22 setMessageBody()

```
Message & Message::setMessageBody (
            const std::string & body )
```

Set the content body of the message.

Parameters

body	The content body.

Returns

The Message object.

3.3.4.23 setMethod()

```
Message & Message::setMethod (
            const std::string & method )
```

Set the HTTP method of the message.

Parameters

method	The HTTP method.

Returns

The Message object.

3.3.4.24 setPort()

```
Message & Message::setPort (
            size_t port )
```

Set the port to which the message is targeted.

Parameters

port	The target port.

Returns

The Message object.

3.3.4.25 setReady()

```
void Message::setReady (
            bool isError )
```

Notify the associated promise/future that the message is completed.

Note that a value of `true` only indicates that the message completed and the response was parsed. It does not indicate anything about the response code of the message (e.g., the status code may be 404, but because the response actually came from the server, this value should be `true`).

A value of `false` indicates that there was a problem either in delivering the message (such as a write failure) or an error in parsing the response.

Parameters

requestCompleted	`true` if this is the result of a successful HTTP request, otherwise `false`.

3.3.4.26 setStatusCode()

```
Message & Message::setStatusCode (
            size_t statusCode )
```

Set the status code of the message.

Per the HTTP spec, this must be a 3-digit number.

Parameters

statusCode	The status code of the message.

Returns

The Message object.

3.3.4.27 setTarget()

```
Message & Message::setTarget (
            const std::string & target )
```

Set the URL target of the message.

Parameters

target	The URL target.

Returns

The Message object.

3.3.4.28 setVersion()

```
Message & Message::setVersion (
            const std::string & version )
```

Set the HTTP version of the message.

Parameters

version	The HTTP version.

Returns

The Message object.

3.3.5 Member Data Documentation

3.3.5.1 headers

```
std::map<std::string, std::vector<std::string> > Ghoti::Wave::Message::headers  [private]
```

A collection of headers and their associated values.

```
headers[field name] = [field value]
```

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

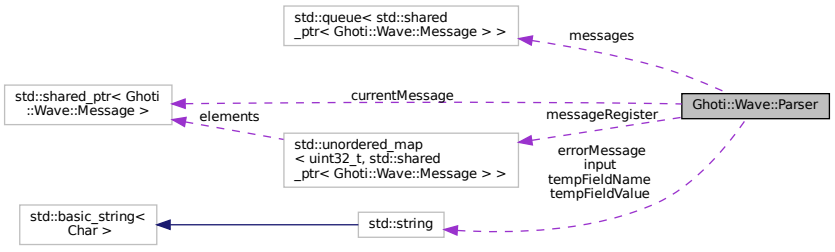
- include/wave/message.hpp
- src/message.cpp

3.4 Ghoti::Wave::Parser Class Reference

Parses a HTTP/1.1 data stream into discrete messages.

```
#include <parser.hpp>
```

Collaboration diagram for Ghoti::Wave::Parser:



Public Types

- enum Type { REQUEST , RESPONSE }

  Represents the type of parsing being performed.

Public Member Functions

- Parser (Type type)

  The constructor.
- void processChunk (const char *buffer, size_t len)

  Process a chunk of data.
- void **parseMessageTarget** (const std::string &target)
- void registerMessage (std::shared_ptr< Message > message)

  Use the provided Message as the recipient of parsing for the Message's id.

Public Attributes

- std::queue< std::shared_ptr< Message > > messages

  A queue of messages that have been parsed so far.

Private Types

- enum ReadStateMajor { NEW_HEADER , FIELD_LINE , MESSAGE_BODY }

  Primary state tracking values.
- enum ReadStateMinor {
  BEGINNING_OF_REQUEST_LINE , BEGINNING_OF_STATUS_LINE , BEGINNING_OF_FIELD_LINE ,
  CRLF ,
  AFTER_CRLF , BEGINNING_OF_REQUEST , BEGINNING_OF_STATUS , METHOD ,
  AFTER_METHOD , REQUEST_TARGET , AFTER_REQUEST_TARGET , HTTP_VERSION ,
  AFTER_HTTP_VERSION , RESPONSE_CODE , REASON_PHRASE , FIELD_NAME ,
  AFTER_FIELD_NAME , BEFORE_FIELD_VALUE , FIELD_VALUE , SINGLETON_FIELD_VALUE ,
  LIST_FIELD_VALUE , UNQUOTED_FIELD_VALUE , QUOTED_FIELD_VALUE_OPEN , QUOTED_FIELD_VALUE_PROCESS
  ,
  QUOTED_FIELD_VALUE_ESCAPE , QUOTED_FIELD_VALUE_CLOSE , AFTER_FIELD_VALUE ,
  FIELD_VALUE_COMMA ,
  AFTER_FIELD_VALUE_COMMA , AFTER_HEADER_FIELDS , MESSAGE_START , MESSAGE_READ }

  Secondary state tracking values.

Private Member Functions

- std::shared_ptr< Message > createNewMessage () const

  Create a new message whose Message::Type matches the Parser::Type of this parser.

Private Attributes

- Type type

  The Parser::Type of HTTP/1.1 stream that will be processed.
- size_t cursor

  An internal counter that indicates the character currently being processed.
- ReadStateMajor readStateMajor

  Tracks the primary state for the parsing state machine.
- ReadStateMinor readStateMinor

  Tracks the secondary state for the parsing state machine.
- size_t majorStart

  Indicates the cursor position at which the major state was last updated.

- size_t minorStart

  Indicates the cursor position at which the minor state was last updated.
- std::string input

  The input string, stored internally so that the stream will be processed correctly, even if it is split across multiple buffered reads.
- std::string errorMessage

  An error message to communicate a parsing issue.
- std::string tempFieldName

  The field name currently being processed.
- std::string tempFieldValue

  The field value currently being processed.
- std::unordered_map< uint32_t, std::shared_ptr< Message > > messageRegister

  A map to store a Message associated with a sequence.
- std::shared_ptr< Message > currentMessage

  The current message being parsed.
- size_t contentLength

  The content length that was encountered when parsing the header.

3.4.1 Detailed Description

Parses a HTTP/1.1 data stream into discrete messages.

3.4.2 Member Enumeration Documentation

3.4.2.1 ReadStateMajor

```
enum Ghoti::Wave::Parser::ReadStateMajor  [private]
```

Primary state tracking values.

These values are used to indicate which major stage the parser is in while parsing the message stream.

The parser uses two stages, to make the parser switch cases easier to follow and to reuse common stages in different contexts (e.g., CRLF).

Enumerator

NEW_HEADER	Expect a new message header.
FIELD_LINE	Expect a new header field.
MESSAGE_BODY	Expect the message body.

3.4.2.2 ReadStateMinor

```
enum Ghoti::Wave::Parser::ReadStateMinor  [private]
```

Secondary state tracking values.

These values are used to indicate which "part" of the primary state is being tracked.

Enumerator

BEGINNING_OF_REQUEST_LINE	A request line is starting.
BEGINNING_OF_STATUS_LINE	A status line is starting.
BEGINNING_OF_FIELD_LINE	A header field line is starting.
CRLF	Expect a CRLF.
AFTER_CRLF	A CRLF has been identified.
BEGINNING_OF_REQUEST	Optional whitespace parsed, request line is now starting.
BEGINNING_OF_STATUS	Optional whitespace parsed, status line is now starting.
METHOD	Method expected.
AFTER_METHOD	Method successfully parsed.
REQUEST_TARGET	Expect request target.
AFTER_REQUEST_TARGET	Request target successfully parsed.
HTTP_VERSION	HTTP version expected.
AFTER_HTTP_VERSION	HTTP version successfully parsed.
RESPONSE_CODE	Response Code Expected.
REASON_PHRASE	Reason Phrase Expected.
FIELD_NAME	Header field name expected.
AFTER_FIELD_NAME	Header field name successfully parsed.
BEFORE_FIELD_VALUE	Header field value about to be processed.
FIELD_VALUE	Header field value expected.
SINGLETON_FIELD_VALUE	Singleton header field value expected.
LIST_FIELD_VALUE	List of header fields expected.
UNQUOTED_FIELD_VALUE	Unquoted field value expected.
QUOTED_FIELD_VALUE_OPEN	Quoted field value begin.
QUOTED_FIELD_VALUE_PROCESS	Quoted field value is being processed.
QUOTED_FIELD_VALUE_ESCAPE	Quoted field value char is being escaped.
QUOTED_FIELD_VALUE_CLOSE	Quoted field value is being closed.
AFTER_FIELD_VALUE	Field value processed.
FIELD_VALUE_COMMA	Field value comma expected.
AFTER_FIELD_VALUE_COMMA	Field value comma processed.
AFTER_HEADER_FIELDS	Header fields processed.
MESSAGE_START	Message started.
MESSAGE_READ	Message being read.

3.4.2.3 Type

`enum Ghoti::Wave::Parser::Type`

Represents the type of parsing being performed.

Enumerator

REQUEST	This is a Request stream.
RESPONSE	This is a Response stream.

3.4.3 Constructor & Destructor Documentation

3.4.3.1 Parser()

```
Parser::Parser (
            Type type )
```

The constructor.

HTTP/1.1 streams do not have an interchangeable syntax, so the stream type must be declared.

The stream will accept an array of bytes, and it will remember its previous parsing position.

Parameters

type	The Parser::Type of the message stream.

3.4.4 Member Function Documentation

3.4.4.1 createNewMessage()

```
shared_ptr< Message > Parser::createNewMessage ( ) const  [private]
```

Create a new message whose Message::Type matches the Parser::Type of this parser.

This function should really only be used by Parser::Type::Request parsing, since all Parser::Type::Response streams should have already registered a Message object to receive the parsed message.

Returns

A properly typed message.

3.4.4.2 processChunk()

```
void Parser::processChunk (
            const char * buffer,
            size_t len )
```

Process a chunk of data.

Parameters

buffer	The buffer to be processed.
len	The length of the buffer in bytes.

3.4.4.3 registerMessage()

```
void Parser::registerMessage (
            std::shared_ptr< Message > message )
```

Use the provided Message as the recipient of parsing for the Message's id.

If a Message with the target ID already exists, then the provided message will adopt the contents of the existing data.

Parameters

message	The object that should receive the desired messages.

3.4.5 Member Data Documentation

3.4.5.1 messageRegister

```
std::unordered_map<uint32_t, std::shared_ptr<Message> > Ghoti::Wave::Parser::messageRegister
[private]
```

A map to store a Message associated with a sequence.

This approach is used so that the parser can be informed of the existence of an expected message. This way, the supplied Message object can act as the recipient of the message as it is parsed.

The registered message should be the same message that was provided to the caller of the Client::sendRequest() function.

```
messageRegister[ID] = message
```

3.4.5.2 messages

```
std::queue<std::shared_ptr<Message> > Ghoti::Wave::Parser::messages
```

A queue of messages that have been parsed so far.

The calling session manager may pop messages from the queue as needed.

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

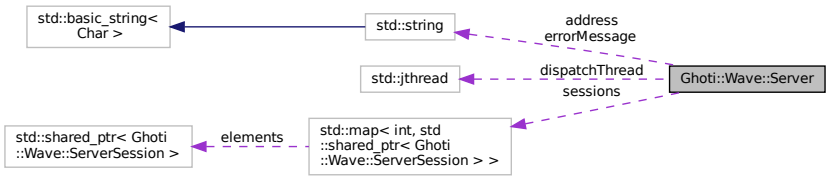
- include/wave/parser.hpp
- src/parser.cpp

3.5 Ghoti::Wave::Server Class Reference

The base Server class.

`#include <server.hpp>`

Collaboration diagram for Ghoti::Wave::Server:



Public Types

- enum ErrorCode { NO_ERROR , SERVER_ALREADY_RUNNING , START_FAILED }

  These are the error codes that the Server may generate when control functions fail.

Public Member Functions

- Server ()

  The constructor.
- ~Server ()

  The destructor.
- Server & clearError ()

  Clears any error code and error message that may be set.
- ErrorCode getErrorCode () const

  Returns the Server::ErrorCode error that was most recently generated.
- const std::string & getErrorMessage () const

  Returns an error message string that was most recently generated.
- bool isRunning () const

  Returns whether or not the server is running.
- Server & setPort (uint16_t port)

  Set the port that the server is listening on.
- uint16_t getPort () const

  Return the server's current port setting.
- Server & setAddress (const char *ip)

  Set the ip address that the server is listening on.
- const std::string & getAddress () const

  Return the server's current ip address setting.
- int getSocketHandle () const

  Returns the socket handle of the server (if set).
- Server & start ()

  Start the server listening on the designated ip address and port.
- Server & stop ()

  Signal the server to stop listening and terminate its thread pool.
- void dispatchLoop (std::stop_token stoken)

  The Dispatch loop used by the thread pool to handle asynchronous reading and writing of the server ports.

Private Attributes

- Ghoti::Pool::Pool workers

  The thread pool worker queue.
- std::map< int, std::shared_ptr< Ghoti::Wave::ServerSession > > sessions

  Stores active sessions.
- std::jthread dispatchThread

  The dispatch thread used to monitor for new connections and to dispatch read/write tasks as needed by the sessions.
- ErrorCode errorCode

  The most recently generated error code.
- std::string errorMessage

  The most recently generated error message.
- bool running

  Stores whether or not the server is set to be running.
- int hSocket

  The socket handle to which the running server is attached.
- std::string address

  The ip address that the server is configured to use.
- uint16_t port

  The port that the server is configured to use.

3.5.1 Detailed Description

The base Server class.

This class is the foundation of the Ghoti.io HTTP server. It serves as the interface to control and expand the server programmatically.

3.5.2 Member Enumeration Documentation

3.5.2.1 ErrorCode

```
enum Ghoti::Wave::Server::ErrorCode
```

These are the error codes that the Server may generate when control functions fail.

Enumerator

NO_ERROR	No error.
SERVER_ALREADY_RUNNING	The change could not be applied because the server is already running.
START_FAILED	The server could not be started.

3.5.3 Constructor & Destructor Documentation

3.5.3.1 Server()

```
Server::Server ( )
```

The constructor.

The constructor only creates the server object. It does not begin listening for connections. In order to begin listening for connections, the Server.start() function must be called.

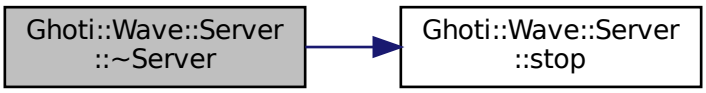
By default, the server will bind to "127.0.0.1" and a port number assigned by the operating system. This default functionality can be changed by using Server.setAddress() and Server.setPort(), respectively.

3.5.3.2 ~Server()

```
Server::~Server ( )
```

The destructor.

The destructor will call Server.stop(). Here is the call graph for this function:



3.5.4 Member Function Documentation

3.5.4.1 clearError()

```
Server & Server::clearError ( )
```

Clears any error code and error message that may be set.

Error messages are not cleared automatically. This function must be called explicitly.

Returns

The server object.

3.5.4.2 dispatchLoop()

```
void Server::dispatchLoop (
            std::stop_token stoken )
```

The Dispatch loop used by the thread pool to handle asynchronous reading and writing of the server ports.

Parameters

stop_token	The stop token provided by the jthread to indicate that the thread should be safely shut down.

3.5.4.3 getAddress()

```
const string & Server::getAddress ( ) const
```

Return the server's current ip address setting.

This setting does not imply that the server is active.

Returns

The current ip address.

3.5.4.4 getErrorCode()

```
Server::ErrorCode Server::getErrorCode ( ) const
```

Returns the Server::ErrorCode error that was most recently generated.

Calling the function does not clear the error. The error must be cleared explicitly by calling Server::clearError().

Returns

The Server::ErrorCode error that was most recently generated.

3.5.4.5 getErrorMessage()

```
const std::string & Server::getErrorMessage ( ) const
```

Returns an error message string that was most recently generated.

Calling the function does not clear the error. The error must be cleared explicitly by calling Server::clearError().

Returns

The error message string that was most recently generated.

3.5.4.6 getPort()

```
uint16_t Server::getPort ( ) const
```

Return the server's current port setting.

This setting does not imply that the server is active.

Returns

The current port number.

3.5.4.7 getSocketHandle()

```
int Server::getSocketHandle ( ) const
```

Returns the socket handle of the server (if set).

Returns

The socket handle of the server.

3.5.4.8 isRunning()

```
bool Server::isRunning ( ) const
```

Returns whether or not the server is running.

Returns

True/False whether or not the server is running.

3.5.4.9 setAddress()

```
Server & Server::setAddress (
            const char * ip )
```

Set the ip address that the server is listening on.

This setting cannot be changed if the server is running. If the server is running, then an error will be set.

Parameters

ip	The ip address that the server should listen on.

Returns

The server object.

3.5.4.10 setPort()

```
Server & Server::setPort (
            uint16_t port )
```

Set the port that the server is listening on.

This setting cannot be changed if the server is running. If the server is running, then an error will be set.

Parameters

port	The port number that the server should listen on.

Returns

The server object.

3.5.4.11 start()

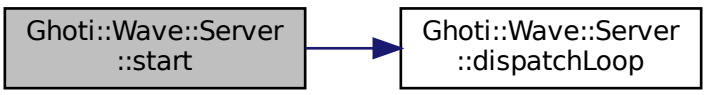
```
Server & Server::start ( )
```

Start the server listening on the designated ip address and port.

Returns

The server object.

Here is the call graph for this function:



3.5.4.12 stop()

```
Server & Server::stop ( )
```

Signal the server to stop listening and terminate its thread pool.

Returns

The server object.

3.5.5 Member Data Documentation

3.5.5.1 sessions

```
std::map<int, std::shared_ptr<Ghoti::Wave::ServerSession> > Ghoti::Wave::Server::sessions
[private]
```

Stores active sessions.

The sessions are keyed by the socket handle to which the session is associated.

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

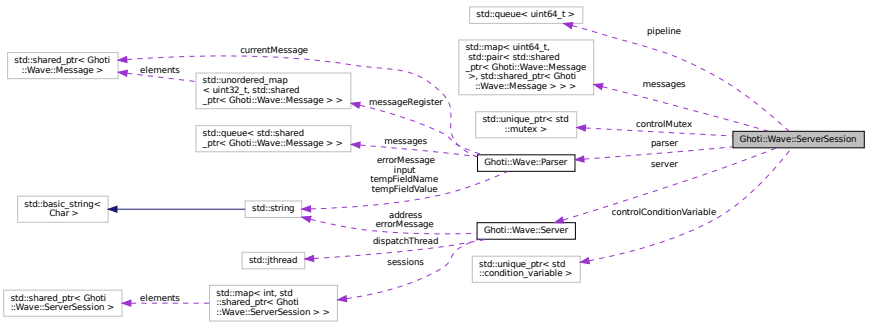
- include/wave/server.hpp
- src/server.cpp

3.6 Ghoti::Wave::ServerSession Class Reference

Represents a persistent connection with a client.

```
#include <serverSession.hpp>
```

Collaboration diagram for Ghoti::Wave::ServerSession:



Public Member Functions

- ServerSession (int hClient, Server *server)

  The constructor.
- ~ServerSession ()

  The destructor.
- bool hasReadDataWaiting ()

  Checks to see whether or not the session has data waiting to be read from the socket.
- bool hasWriteDataWaiting ()

  Checks to see whether or not the session has data waiting to be written to the socket.
- bool isFinished ()

  Indicates whether or not the session has completed all communications and may be terminated.
- void read ()

  Perform a read from the session.
- void write ()

  Perform a write to the session.

Public Attributes

- std::unique_ptr< std::mutex > controlMutex

  Used to synchronize access to the session to make it thread safe.
- std::unique_ptr< std::condition_variable > controlConditionVariable

  Used to synchronize access to the session to make it thread safe.

Private Attributes

- int hClient

  The socket handle to the client.
- size_t requestSequence

  A monotonically increasing counter to track request/response pairs.
- size_t writeOffset

  A byte offset used to track how many bytes of a message have been written, so that individual write attempts do not duplicate data.
- bool working

  Tracks whether or not the session has work queued.
- bool finished

  Tracks whether or not the session has completed all pending communications.
- Parser parser

  The parser object used to parse the raw HTTP stream.
- Server * server

  A pointer to the server object.
- std::map< uint64_t, std::pair< std::shared_ptr< Message >, std::shared_ptr< Message > > > messages

  Tracks request/response pairs.
- std::queue< uint64_t > pipeline

  Simple queue to track which request sequence # should be parsed next.

3.6.1 Detailed Description

Represents a persistent connection with a client.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 ServerSession()

```
ServerSession::ServerSession (
            int hClient,
            Server * server )
```

The constructor.

Parameters

hClient	The socket handle to the client connection.
server	A pointer to the parent Server object.

3.6.3 Member Function Documentation

3.6.3.1 hasReadDataWaiting()

```
bool ServerSession::hasReadDataWaiting ( )
```

Checks to see whether or not the session has data waiting to be read from the socket.

This is non-blocking mutex controlled. If the session is currently working, then this function will return false.

Returns

Whether or not the session has data waiting to be read from the socket.

3.6.3.2 hasWriteDataWaiting()

```
bool ServerSession::hasWriteDataWaiting ( )
```

Checks to see whether or not the session has data waiting to be written to the socket.

This is non-blocking mutex controlled. If the session is currently working, then this function will return false.

Returns

Whether or not the session has data waiting to be written to the socket.

3.6.3.3 isFinished()

```
bool ServerSession::isFinished ( )
```

Indicates whether or not the session has completed all communications and may be terminated.

Returns

`true` if all communications have completed, `false` otherwise.

3.6.3.4 read()

```
void ServerSession::read ( )
```

Perform a read from the session.

This function is intended to be called by the server's thread pool worker queue, probably in a lambda expression.

3.6.3.5 write()

```
void ServerSession::write ( )
```

Perform a write to the session.

This function is intended to be called by the server's thread pool worker queue, probably in a lambda expression.

3.6.4 Member Data Documentation

3.6.4.1 messages

```
std::map<uint64_t, std::pair<std::shared_ptr<Message>, std::shared_ptr<Message> > > Ghoti↩
::Wave::ServerSession::messages  [private]
```

Tracks request/response pairs.

`messages[request sequence #] = <request, response>`

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

- include/wave/serverSession.hpp
- src/serverSession.cpp

Chapter 4

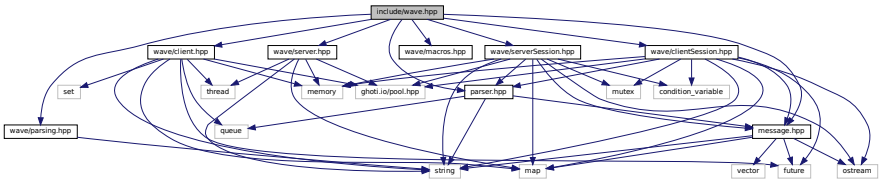
File Documentation

4.1 include/wave.hpp File Reference

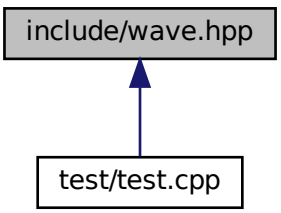
Header file supplied for use by 3rd party code so that they can easily include all necessary headers for the Ghoti.io Wave library.

```
#include "wave/client.hpp"
#include "wave/clientSession.hpp"
#include "wave/macros.hpp"
#include "wave/message.hpp"
#include "wave/parser.hpp"
#include "wave/parsing.hpp"
#include "wave/server.hpp"
#include "wave/serverSession.hpp"
```

Include dependency graph for wave.hpp:



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



4.1.1 Detailed Description

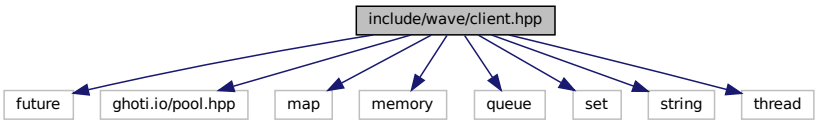
Header file supplied for use by 3rd party code so that they can easily include all necessary headers for the Ghoti.io Wave library.

4.2 include/wave/client.hpp File Reference

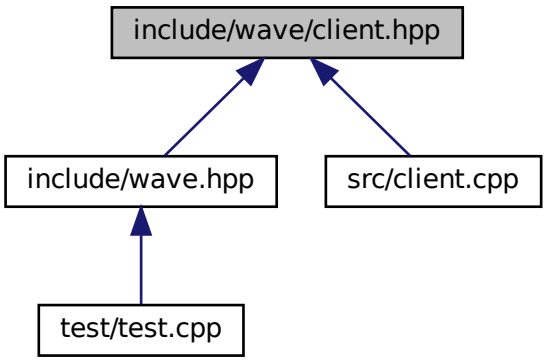
Header file for declaring the Client class.

```
#include <future>
#include <ghoti.io/pool.hpp>
#include <map>
#include <memory>
#include <queue>
#include <set>
#include <string>
#include <thread>
```

Include dependency graph for client.hpp:



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



Classes

- class Ghoti::Wave::Client

  Represents a client and all of its HTTP connections.

4.2.1 Detailed Description

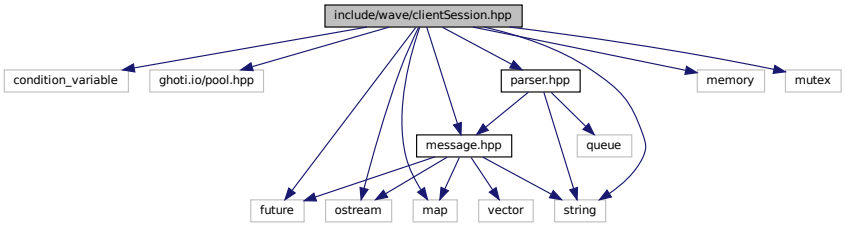
Header file for declaring the Client class.

4.3 include/wave/clientSession.hpp File Reference

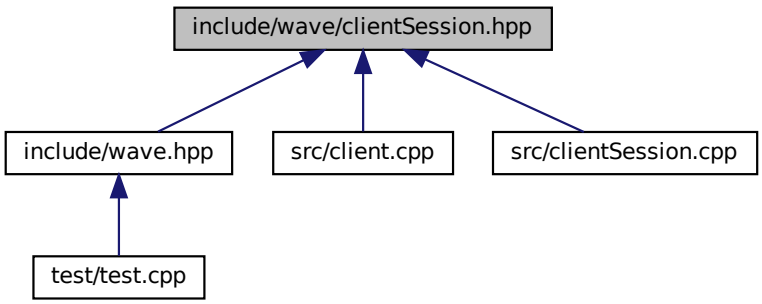
Header file for declaring the ClientSession class.

```
#include <condition_variable>
#include <ghoti.io/pool.hpp>
#include <future>
#include <memory>
#include <mutex>
#include <ostream>
#include "parser.hpp"
#include <map>
#include "message.hpp"
#include <string>
```

Include dependency graph for clientSession.hpp:



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



Classes

- class Ghoti::Wave::ClientSession

  Represents a connection to a particular domain/port pair.

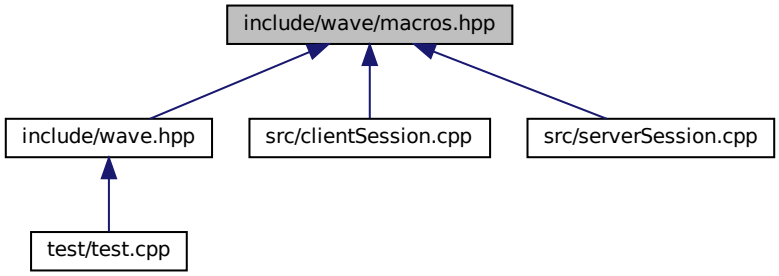
4.3.1 Detailed Description

Header file for declaring the ClientSession class.

4.4 include/wave/macros.hpp File Reference

Header file for declaring the Client class.

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



Variables

- constexpr size_t **Ghoti::Wave::MAXBUFFERSIZE** = 4096

4.4.1 Detailed Description

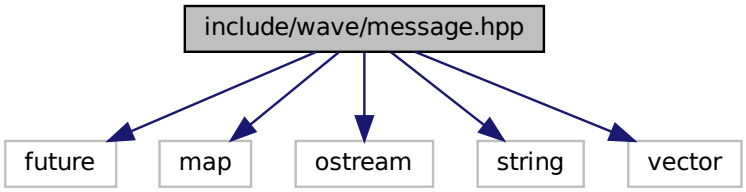
Header file for declaring the Client class.

4.5 include/wave/message.hpp File Reference

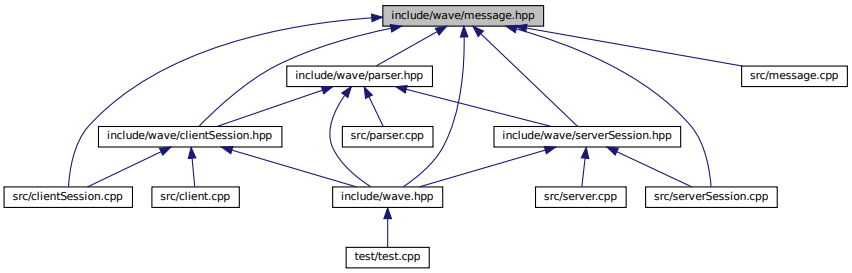
Header file for declaring the Message class.

```
#include <future>
#include <map>
#include <ostream>
#include <string>
```

`#include <vector>`
Include dependency graph for message.hpp:



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



Classes

- class Ghoti::Wave::Message
  Represents a HTTP message.

Functions

- std::ostream & Ghoti::Wave::operator<< (std::ostream &out, Message &message)
  Helper function to output a Message to a stream.

4.5.1 Detailed Description

Header file for declaring the Message class.

4.5.2 Function Documentation

4.5.2.1 operator<<()

```
ostream & Ghoti::Wave::operator<< (
            std::ostream & out,
            Message & message )
```

Helper function to output a Message to a stream.

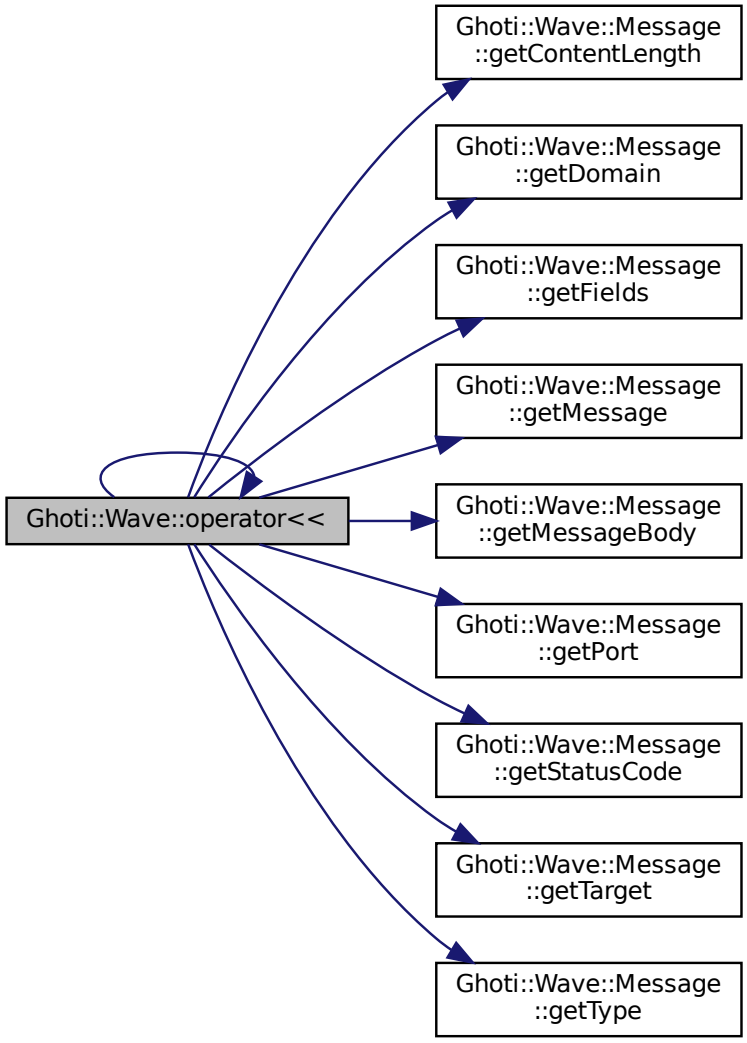
Parameters

out	The output stream.
message	The Message to be inserted into the stream.

Returns

The output stream.

Here is the call graph for this function:

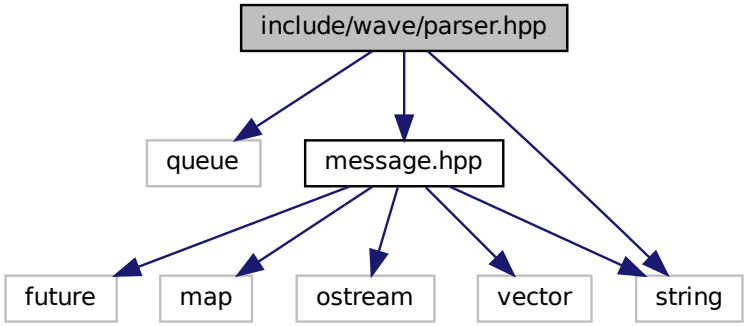


4.6 include/wave/parser.hpp File Reference

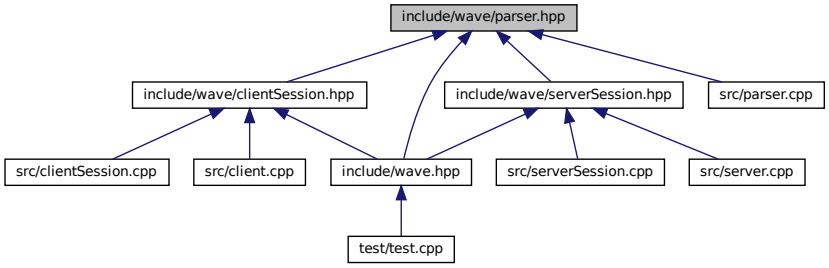
Header file for declaring the Session class.

```
#include <queue>
#include "message.hpp"
#include <string>
```

Include dependency graph for parser.hpp:



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



Classes

- class Ghoti::Wave::Parser

  Parses a HTTP/1.1 data stream into discrete messages.

4.6.1 Detailed Description

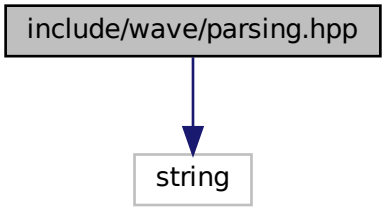
Header file for declaring the Session class.

4.7 include/wave/parsing.hpp File Reference

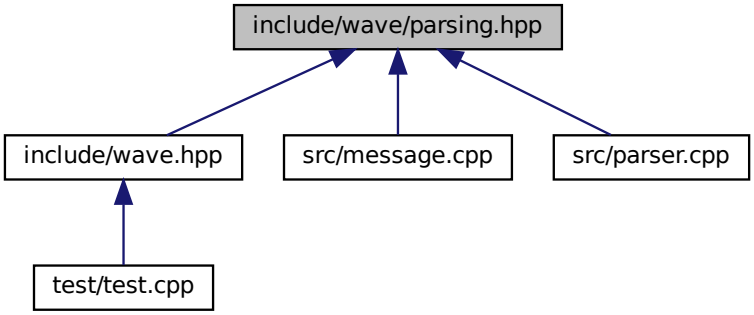
Header file for declaring text parsing functions.

```
#include <string>
```

Include dependency graph for parsing.hpp:



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



Functions

- bool Ghoti::Wave::isListField (const std::string &name)

  Identify a field name as accepting a list-based set of values.
- bool Ghoti::Wave::isTokenChar (uint8_t c)

  Identify valid Token characters.
- bool Ghoti::Wave::isWhitespaceChar (uint8_t c)

  Identify valid whitespace characters.
- bool Ghoti::Wave::isVisibleChar (uint8_t c)

  Identify valid Visible (printing) characters.
- bool Ghoti::Wave::isObsoleteTextChar (uint8_t c)

  Identify valid obs-text characters.

- bool Ghoti::Wave::isFieldNameChar (uint8_t c)

  Identify valid field-name characters.
- bool Ghoti::Wave::isQuotedChar (uint8_t c)

  Identify valid quoted characters.
- bool Ghoti::Wave::isFieldContentChar (uint8_t c)

  Identify valid field-content characters.
- bool Ghoti::Wave::isCRLFChar (uint8_t c)

  Identify CRLF characters.
- bool Ghoti::Wave::fieldValueQuotesNeeded (const std::string &str)

  Indicate whether or not the string contains a character which makes it necessary to wrap the string in double quotes.
- std::string Ghoti::Wave::fieldValueEscape (const std::string &str)

  Escape a field value.

4.7.1 Detailed Description

Header file for declaring text parsing functions.

4.7.2 Function Documentation

4.7.2.1 fieldValueEscape()

```
string Ghoti::Wave::fieldValueEscape (
            const std::string & str )
```

Escape a field value.

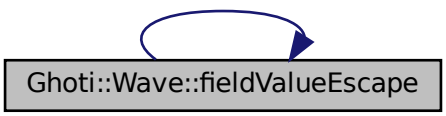
Parameters

str	The field value to be escaped.

Returns

The escaped field value.

Here is the call graph for this function:



4.7.2.2 fieldValueQuotesNeeded()

```
bool Ghoti::Wave::fieldValueQuotesNeeded (
            const std::string & str )
```

Indicate whether or not the string contains a character which makes it necessary to wrap the string in double quotes.

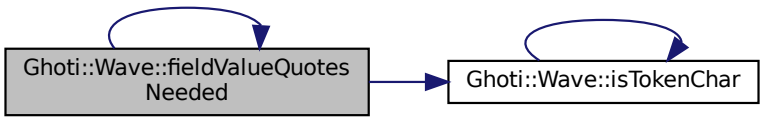
Parameters

str	The string in question.

Returns

Whether or not the string needs to be wrapped in double quotes.

Here is the call graph for this function:



4.7.2.3 isCRLFChar()

```
bool Ghoti::Wave::isCRLFChar (
            uint8_t c )
```

Identify CRLF characters.

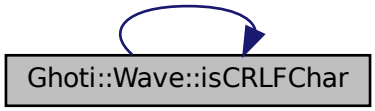
Parameters

c	The character to test.

Returns

Whether or not the character is a valid CRLF character.

Here is the call graph for this function:



4.7.2.4 isFieldContentChar()

```
bool Ghoti::Wave::isFieldContentChar (
            uint8_t c )
```

Identify valid field-content characters.

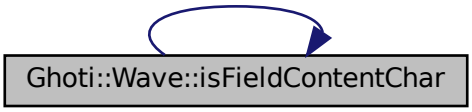
Parameters

c	The character to test.

Returns

Whether or not the character is a valid field-content character.

Here is the call graph for this function:



4.7.2.5 isFieldNameChar()

```
bool Ghoti::Wave::isFieldNameChar (
            uint8_t c )
```

Identify valid field-name characters.

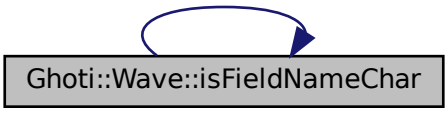
Parameters

c	The character to test.

Returns

Whether or not the character is a valid field-name character.

Here is the call graph for this function:



4.7.2.6 isListField()

```
bool Ghoti::Wave::isListField (
            const std::string & name )
```

Identify a field name as accepting a list-based set of values.

Parameters

name	The field name. The field name must be uppercase.

Returns

Whether or not the field name is recognized as a list-based field.

Here is the call graph for this function:

4.7.2.7 isObsoleteTextChar()

```
bool Ghoti::Wave::isObsoleteTextChar (
            uint8_t c )
```

Identify valid obs-text characters.

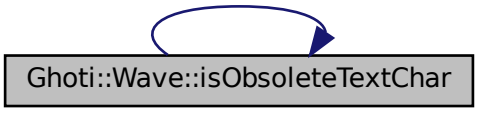
Parameters

c	The character to test.

Returns

Whether or not the character is a valid obs-text character.

Here is the call graph for this function:



4.7.2.8 isQuotedChar()

```
bool Ghoti::Wave::isQuotedChar (
            uint8_t c )
```

Identify valid quoted characters.

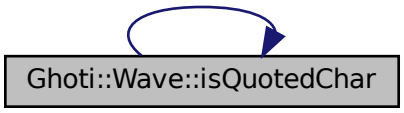
Parameters

c	The character to test.

Returns

Whether or not the character is a valid quoted character.

Here is the call graph for this function:



4.7.2.9 isTokenChar()

```
bool Ghoti::Wave::isTokenChar (
            uint8_t c )
```

Identify valid Token characters.

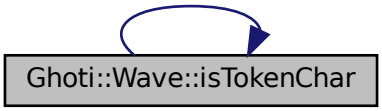
Parameters

c	The character to test.

Returns

Whether or not the character is a valid token character.

Here is the call graph for this function:



4.7.2.10 isVisibleChar()

```
bool Ghoti::Wave::isVisibleChar (
            uint8_t c )
```

Identify valid Visible (printing) characters.

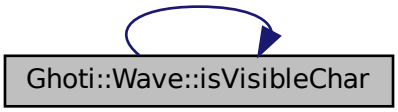
Parameters

c	The character to test.

Returns

Whether or not the character is a valid visible character.

Here is the call graph for this function:



4.7.2.11 isWhitespaceChar()

```
bool Ghoti::Wave::isWhitespaceChar (
            uint8_t c )
```

Identify valid whitespace characters.

Parameters

c	The character to test.

Returns

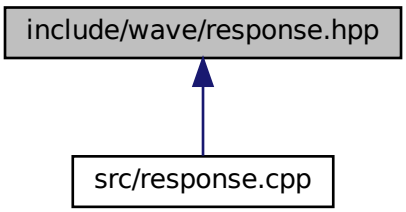
Whether or not the character is a valid visible character.

Here is the call graph for this function:

4.8 include/wave/response.hpp File Reference

Header file for declaring the Response class.

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



4.8.1 Detailed Description

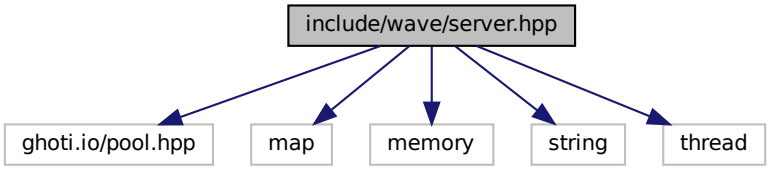
Header file for declaring the Response class.

4.9 include/wave/server.hpp File Reference

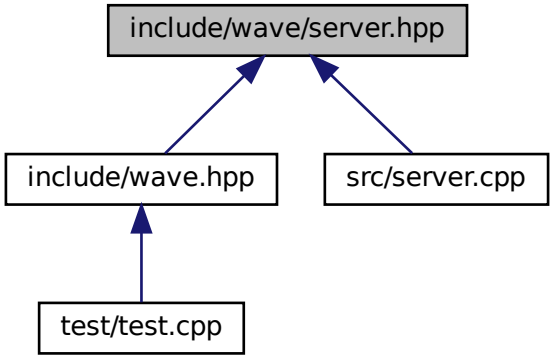
Header file for declaring the Server class.

```
#include <ghoti.io/pool.hpp>
#include <map>
#include <memory>
#include <string>
#include <thread>
```

Include dependency graph for server.hpp:



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



Classes

- class Ghoti::Wave::Server

  The base Server class.

4.9.1 Detailed Description

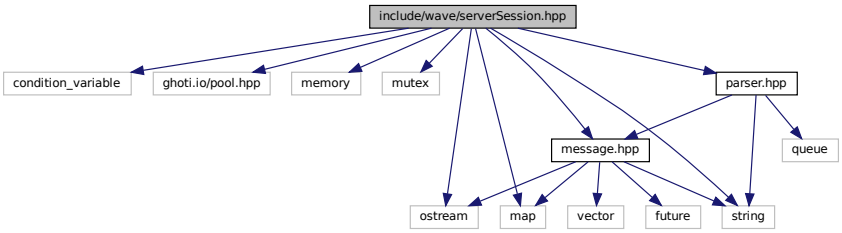
Header file for declaring the Server class.

4.10 include/wave/serverSession.hpp File Reference

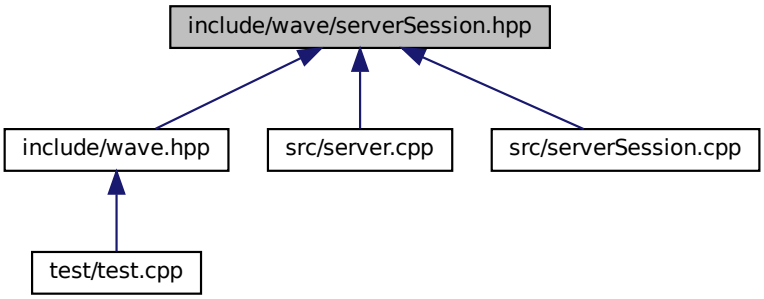
Header file for declaring the ServerSession class.

```
#include <condition_variable>
#include <ghoti.io/pool.hpp>
#include <memory>
#include <mutex>
#include <ostream>
#include "parser.hpp"
#include <map>
#include "message.hpp"
#include <string>
```

Include dependency graph for serverSession.hpp:



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



Classes

- class Ghoti::Wave::ServerSession

  Represents a persistent connection with a client.

4.10.1 Detailed Description

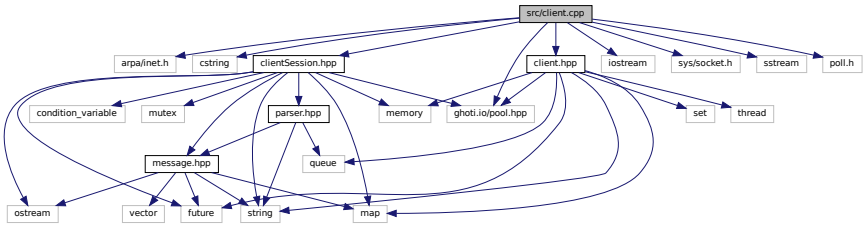
Header file for declaring the ServerSession class.

4.11 src/client.cpp File Reference

Define the Ghoti::Wave::Client class.

```
#include <arpa/inet.h>
#include <cstring>
#include <ghoti.io/pool.hpp>
#include <iostream>
#include <sys/socket.h>
#include <sstream>
#include <poll.h>
#include "client.hpp"
#include "clientSession.hpp"
```

Include dependency graph for client.cpp:



4.11.1 Detailed Description

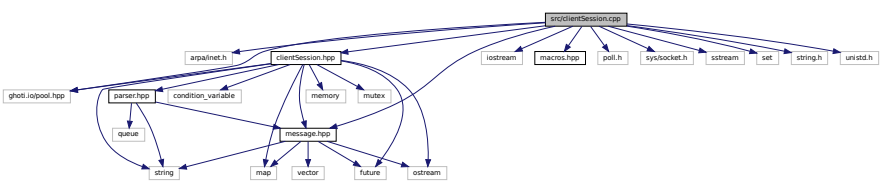
Define the Ghoti::Wave::Client class.

4.12 src/clientSession.cpp File Reference

Define the Ghoti::Wave::ClientSession class.

```
#include <arpa/inet.h>
#include "clientSession.hpp"
#include <ghoti.io/pool.hpp>
#include <iostream>
#include "macros.hpp"
#include "message.hpp"
#include <poll.h>
#include <sys/socket.h>
#include <sstream>
#include <set>
#include <string.h>
#include <unistd.h>
```

Include dependency graph for clientSession.cpp:



4.12.1 Detailed Description

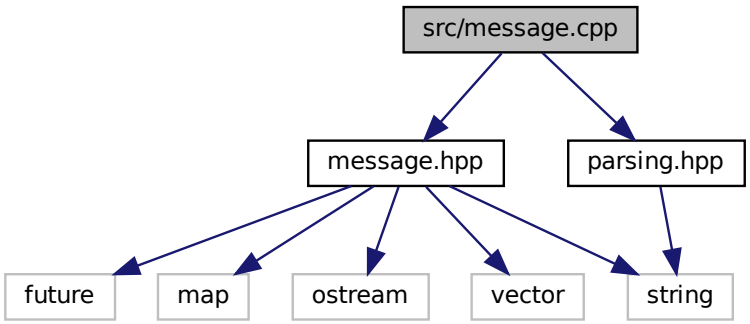
Define the Ghoti::Wave::ClientSession class.

4.13 src/message.cpp File Reference

Define the Ghoti::Wave::Message class.

```
#include "message.hpp"
#include "parsing.hpp"
```

Include dependency graph for message.cpp:



4.13.1 Detailed Description

Define the Ghoti::Wave::Message class.

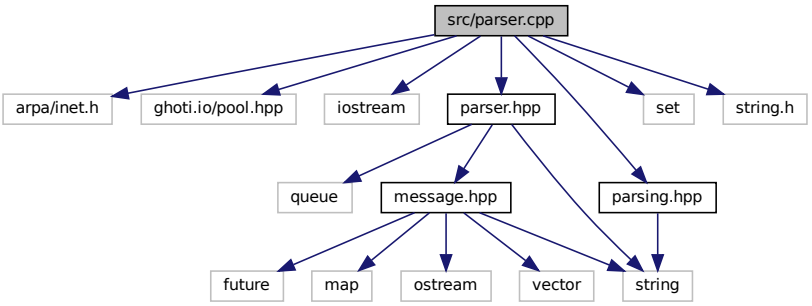
4.14 src/parser.cpp File Reference

Define the Ghoti::Wave::Parser class.

```
#include <arpa/inet.h>
#include <ghoti.io/pool.hpp>
#include <iostream>
#include "parser.hpp"
#include "parsing.hpp"
#include <set>
```

```
#include <string.h>
```

Include dependency graph for parser.cpp:



Macros

- #define **SET_NEW_HEADER**
- #define **SET_MINOR_STATE**(nextState)
- #define **SET_MAJOR_STATE**(nextMajorState, nextMinorState)
- #define **READ_WHITESPACE_OPTIONAL**(nextState)
- #define **READ_WHITESPACE_REQUIRED**(nextState, statusCode, errorMessage)
- #define **READ_CRLF_OPTIONAL**(nextState)
- #define **READ_CRLF_REQUIRED**(nextState, statusCode, errorMessage)
- #define **REQUEST_STATUS_ERROR** (this->type == REQUEST ? "Error reading request line." : "Error reading status line.")

4.14.1 Detailed Description

Define the Ghoti::Wave::Parser class.

4.14.2 Macro Definition Documentation

4.14.2.1 READ_CRLF_OPTIONAL

```
#define READ_CRLF_OPTIONAL(
              nextState )
```

Value:

```
  while (this->cursor < input_length) { \
    if ((this->input[this->cursor] != '\r') && (this->input[this->cursor] != '\n')) { \
      SET_MINOR_STATE(nextState); \
      break; \
    } \
    ++this->cursor; \
  }
```

4.14.2.2 READ_CRLF_REQUIRED

```
#define READ_CRLF_REQUIRED(
              nextState,
              statusCode,
              errorMessage )
```

Value:

```
  size_t len = this->cursor - this->minorStart; \
  while ((this->cursor < input_length) && (len < 2)) { \
    if (((len == 0) && !((this->input[this->cursor] == '\r') || (this->input[this->cursor] == '\n'))) \
      || ((len == 1) && (this->input[this->cursor] != '\n'))) { \
      this->currentMessage->setStatusCode(statusCode).setErrorMessage(errorMessage); \
    } \
    if (!this->currentMessage->hasError() && (this->input[this->cursor] == '\n')) { \
      SET_MINOR_STATE(nextState); \
      ++this->cursor; \
      break; \
    } \
    ++this->cursor; \
    ++len; \
  }
```

4.14.2.3 READ_WHITESPACE_OPTIONAL

```
#define READ_WHITESPACE_OPTIONAL(
              nextState )
```

Value:

```
  while ((this->cursor < input_length) && ( \
      isspace(this->input[this->cursor]) \
      && (this->input[this->cursor] != '\n') \
      && (this->input[this->cursor] != '\r'))) { \
    ++this->cursor; \
  } \
  if ((this->cursor < input_length) && ( \
      !isspace(this->input[this->cursor]) \
      || (this->input[this->cursor] == '\n') \
      || (this->input[this->cursor] == '\r'))) { \
    SET_MINOR_STATE(nextState); \
  }
```

4.14.2.4 READ_WHITESPACE_REQUIRED

```
#define READ_WHITESPACE_REQUIRED(
              nextState,
              statusCode,
              errorMessage )
```

Value:

```
  while ((this->cursor < input_length) && ( \
      isspace(this->input[this->cursor]) \
      && (this->input[this->cursor] != '\n') \
      && (this->input[this->cursor] != '\r'))) { \
    ++this->cursor; \
  } \
  if (this->cursor < input_length) { \
    if (this->cursor > this->minorStart) { \
      SET_MINOR_STATE(nextState); \
    } \
    else { \
      this->currentMessage->setStatusCode(statusCode).setErrorMessage(errorMessage); \
    } \
  }
```

4.14.2.5 SET_MAJOR_STATE

```
#define SET_MAJOR_STATE(
              nextMajorState,
              nextMinorState )
```

Value:

```
  this->readStateMajor = nextMajorState; \
  this->majorStart = this->cursor; \
  SET_MINOR_STATE(nextMinorState);
```

4.14.2.6 SET_MINOR_STATE

```
#define SET_MINOR_STATE(
              nextState )
```

Value:

```
  this->readStateMinor = nextState; \
  this->minorStart = this->cursor;
```

4.14.2.7 SET_NEW_HEADER

```
#define SET_NEW_HEADER
```

Value:

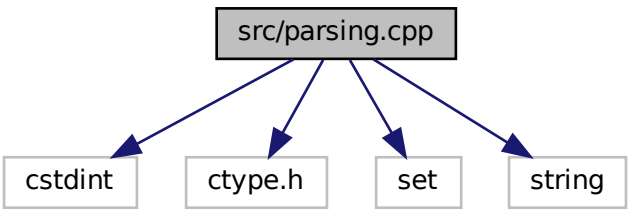
```
  this->readStateMajor = NEW_HEADER; \
  this->readStateMinor = this->type == REQUEST \
    ? BEGINNING_OF_REQUEST_LINE \
    : BEGINNING_OF_STATUS_LINE; \
  this->majorStart = this->cursor; \
  this->minorStart = this->cursor; \
  this->contentLength = 0;
```

4.15 src/parsing.cpp File Reference

Define the text parsing functions.

```
#include <cstdint>
#include <ctype.h>
#include <set>
#include <string>
```

Include dependency graph for parsing.cpp:



Functions

- bool Ghoti::Wave::isListField (const std::string &name)

  Identify a field name as accepting a list-based set of values.
- bool Ghoti::Wave::isTokenChar (uint8_t c)

  Identify valid Token characters.
- bool Ghoti::Wave::isWhitespaceChar (uint8_t c)

  Identify valid whitespace characters.
- bool Ghoti::Wave::isVisibleChar (uint8_t c)

  Identify valid Visible (printing) characters.
- bool Ghoti::Wave::isObsoleteTextChar (uint8_t c)

  Identify valid obs-text characters.
- bool Ghoti::Wave::isFieldNameChar (uint8_t c)

  Identify valid field-name characters.
- bool Ghoti::Wave::isQuotedChar (uint8_t c)

  Identify valid quoted characters.
- bool Ghoti::Wave::isFieldContentChar (uint8_t c)

  Identify valid field-content characters.
- bool Ghoti::Wave::isCRLFChar (uint8_t c)

  Identify CRLF characters.
- bool Ghoti::Wave::fieldValueQuotesNeeded (const std::string &str)

  Indicate whether or not the string contains a character which makes it necessary to wrap the string in double quotes.
- std::string Ghoti::Wave::fieldValueEscape (const std::string &str)

  Escape a field value.

4.15.1 Detailed Description

Define the text parsing functions.

4.15.2 Function Documentation

4.15.2.1 fieldValueEscape()

```
string Ghoti::Wave::fieldValueEscape (
            const std::string & str )
```

Escape a field value.

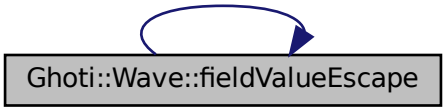
Parameters

str	The field value to be escaped.

Returns

The escaped field value.

Here is the call graph for this function:



4.15.2.2 fieldValueQuotesNeeded()

```
bool Ghoti::Wave::fieldValueQuotesNeeded (
            const std::string & str )
```

Indicate whether or not the string contains a character which makes it necessary to wrap the string in double quotes.

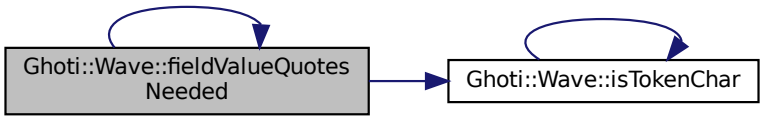
Parameters

str	The string in question.

Returns

Whether or not the string needs to be wrapped in double quotes.

Here is the call graph for this function:



4.15.2.3 isCRLFChar()

```
bool Ghoti::Wave::isCRLFChar (
            uint8_t c )
```

Identify CRLF characters.

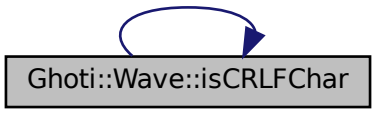
Parameters

c	The character to test.

Returns

Whether or not the character is a valid CRLF character.

Here is the call graph for this function:



4.15.2.4 isFieldContentChar()

```
bool Ghoti::Wave::isFieldContentChar (
            uint8_t c )
```

Identify valid field-content characters.

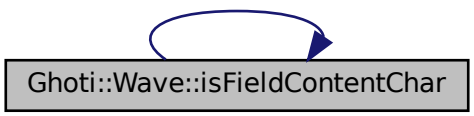
Parameters

c	The character to test.

Returns

Whether or not the character is a valid field-content character.

Here is the call graph for this function:



4.15.2.5 isFieldNameChar()

```
bool Ghoti::Wave::isFieldNameChar (
            uint8_t c )
```

Identify valid field-name characters.

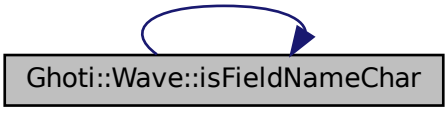
Parameters

c	The character to test.

Returns

Whether or not the character is a valid field-name character.

Here is the call graph for this function:



4.15.2.6 isListField()

```
bool Ghoti::Wave::isListField (
            const std::string & name )
```

Identify a field name as accepting a list-based set of values.

Parameters

name	The field name. The field name must be uppercase.

Returns

Whether or not the field name is recognized as a list-based field.

Here is the call graph for this function:

4.15.2.7 isObsoleteTextChar()

```
bool Ghoti::Wave::isObsoleteTextChar (
            uint8_t c )
```

Identify valid obs-text characters.

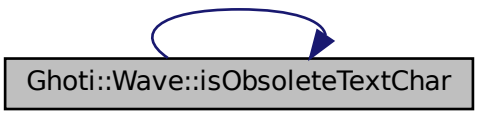
Parameters

c	The character to test.

Returns

Whether or not the character is a valid obs-text character.

Here is the call graph for this function:



4.15.2.8 isQuotedChar()

```
bool Ghoti::Wave::isQuotedChar (
            uint8_t c )
```

Identify valid quoted characters.

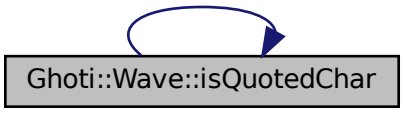
Parameters

c	The character to test.

Returns

Whether or not the character is a valid quoted character.

Here is the call graph for this function:



4.15.2.9 isTokenChar()

```
bool Ghoti::Wave::isTokenChar (
            uint8_t c )
```

Identify valid Token characters.

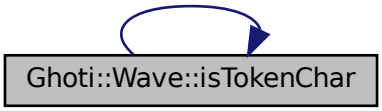
Parameters

c	The character to test.

Returns

Whether or not the character is a valid token character.

Here is the call graph for this function:



4.15.2.10 isVisibleChar()

```
bool Ghoti::Wave::isVisibleChar (
            uint8_t c )
```

Identify valid Visible (printing) characters.

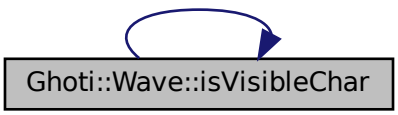
Parameters

c	The character to test.

Returns

Whether or not the character is a valid visible character.

Here is the call graph for this function:



4.15.2.11 isWhitespaceChar()

```
bool Ghoti::Wave::isWhitespaceChar (
            uint8_t c )
```

Identify valid whitespace characters.

Parameters

c	The character to test.

Returns

Whether or not the character is a valid visible character.

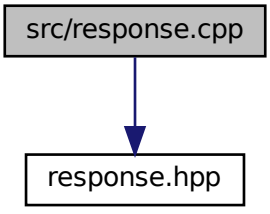
Here is the call graph for this function:

4.16 src/response.cpp File Reference

Define the Ghoti::Wave::Response class.

```
#include "response.hpp"
```

Include dependency graph for response.cpp:



4.16.1 Detailed Description

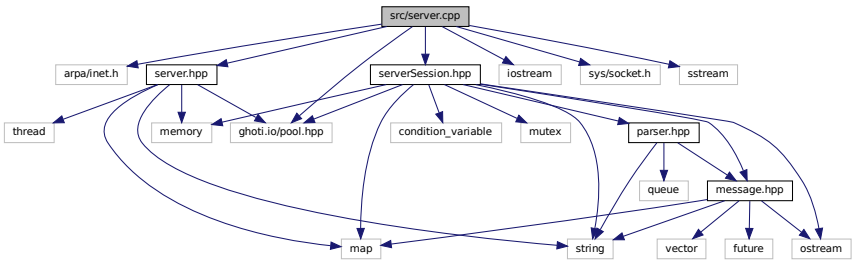
Define the Ghoti::Wave::Response class.

4.17 src/server.cpp File Reference

Define the Ghoti::Wave::Server class.

```
#include <arpa/inet.h>
#include <ghoti.io/pool.hpp>
#include <iostream>
#include <sys/socket.h>
#include <sstream>
#include "server.hpp"
#include "serverSession.hpp"
```

Include dependency graph for server.cpp:



4.17.1 Detailed Description

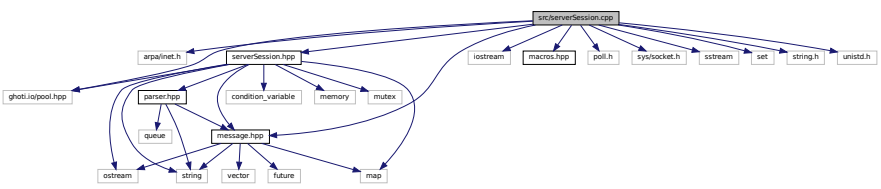
Define the Ghoti::Wave::Server class.

4.18 src/serverSession.cpp File Reference

Define the Ghoti::Wave::ServerSession class.

```
#include <arpa/inet.h>
#include <ghoti.io/pool.hpp>
#include <iostream>
#include "macros.hpp"
#include "message.hpp"
#include <poll.h>
#include <sys/socket.h>
#include <sstream>
#include <set>
#include "serverSession.hpp"
#include <string.h>
#include <unistd.h>
```

Include dependency graph for serverSession.cpp:



4.18.1 Detailed Description

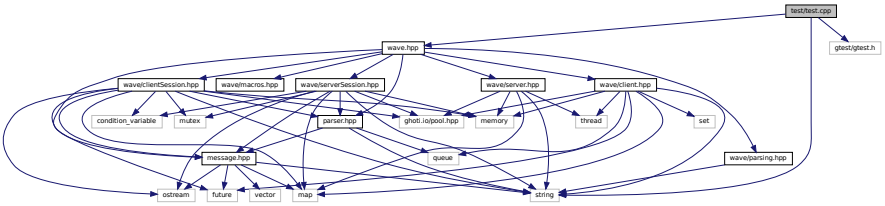
Define the Ghoti::Wave::ServerSession class.

4.19 test/test.cpp File Reference

Test the general Wave server behavior.

```
#include <string>
#include <gtest/gtest.h>
#include "wave.hpp"
```

Include dependency graph for test.cpp:



Functions

- **TEST** (Server, Startup)
- int **main** (int argc, char ∗∗argv)

Variables

- Server **s** {}
- uint16_t **serverPort** = 0
- constexpr auto **quantum** {10ms}

4.19.1 Detailed Description

Test the general Wave server behavior.

Index