Wrench Documentation

Release 2.0.0-beta

Dominic Scheirlinck and Contributors

Aug 15, 2017

Contents

Wrench is a WebSockets library for PHP 7.1

CHAPTER 1

Introduction

Wrench is a simple websocket server and client package for PHP 7.1

php-websocket

Wrench was previously known as php-websocket. Why the name change? See Frequently Asked Questions about the PHP License. Also, the namespace WebSocket is too generic; it denotes a common functionality, and may already be in use by application code. The BC break of a new major version was a good time to introduce this move to best practices.

CHAPTER 2

Installing Wrench

The library is PSR-4 compatible, with a vendor name of **Wrench**.

composer

Wrench is available on Packagist as wrench/wrench.

Here's what it looks like in your `composer.json`

```
{
    ...
    "require": {
        "wrench/wrench": "~3.0"
    }
}
```

CHAPTER 3

Getting Started

Starting a Server

The first thing you'll want to do to serve WebSockets from PHP is start a WebSockets server. Wrench provides a simple Server class that implements the most recent version of the WebSockets protocol. Subclassing the Server class is encouraged: see WebSocketBasicServer for an example.

When you're ready for your server to start responding to requests, call $server->run():

```
use Wrench\BasicServer;

$server = new BasicServer('ws://localhost:8000', array(
    'allowed_origins' => array(
        'mysite.com',
        'mysite.dev.localdomain'
    )
));

// Logging is via PSR3
$logger = new Monolog\Logger('name');
$server->setLogger($logger);

// Register your applications here
$server->registerApplication('echo', new \Wrench\Examples\EchoApplication());
$server->registerApplication('chat', new \My\ChatApplication());

$server->run();
```

CHAPTER 4

Performance

Wrench uses a single-process server, without threads, and blocks while processing data from any client. This means it has little hope of scaling in production.

You might like to use some middleware between your PHP application code and WebSocket clients in production. For example, you might use something like RabbitMQ's STOMP + WebSockets Plugin. In any case, if you're hoping to serve large numbers of clients, you should probably look into one of the evented IO based servers.

CHAPTER 5

API Documentation

Wrench\Application

Wrench\Application\Application

class **Application**
Wrench Server Application

onData (*$payload*, *$connection*)
Handle data received from a client

Parameters

- **$payload** (Payload) – A payload object, that supports __toString()
- **$connection** (Connection) –

Wrench\Application\EchoApplication

class **EchoApplication**
Example application for Wrench: echo server

onData (*$data*, *$client*)

Parameters

- **$data** –
- **$client** –

Wrench\BasicServer

class `BasicServer`

constant `EVENT_SOCKET_CONNECT`
Events

property `rateLimiter`
protected

property `originPolicy`
protected

property `uri`
protected string

The URI of the server

property `options`
protected array

Options

property `logger`
protected Closure

A logging callback

The default callback simply prints to stdout. You can pass your own logger in the options array. It should take a string message and string priority as parameters.

property `listeners`
protected array<string

Event listeners

Add listeners using the addListener() method.

property `connectionManager`
protected ConnectionManager

Connection manager

property `applications`
protected array<string

Applications

property `protocol`
protected Protocol

`__construct` (*$uri*, *$options = array()*)
Constructor

Parameters

- **`$uri`** (*string*) –
- **`$options`** (*array*) –

`configure` (*$options*)

Parameters

- **`$options`** –

configureRateLimiter()

configureOriginPolicy()
Configures the origin policy

addAllowedOrigin(*$origin*)
Adds an allowed origin

Parameters

- **$origin** (*array*) –

configureLogger()
Configures the logger

Returns void

configureConnectionManager()
Configures the connection manager

Returns void

getConnectionManager()
Gets the connection manager

Returns WrenchConnectionManager

getUri()

Returns string

setLogger(*$logger*)
Sets a logger

Parameters

- **$logger** (*Closure*) –

Returns void

run()
Main server loop

Returns void This method does not return!

log(*$message*, *$priority = 'info'*)
Logs a message to the server log

The default logger simply prints the message to stdout. You can provide a logging closure. This is useful, for instance, if you've daemonized and closed STDOUT.

Parameters

- **$message** (*string*) – Message to display.
- **$priority** –

Returns void

notify(*$event*, *$arguments = array()*)
Notifies listeners of an event

Parameters

- **$event** (*string*) –
- **$arguments** (*array*) – Event arguments

Returns void

addListener(*$event*, *$callback*)
Adds a listener

Provide an event (see the Server::EVENT_* constants) and a callback closure. Some arguments may be provided to your callback, such as the connection the caused the event.

Parameters

- **$event** (*string*) –
- **$callback** (*Closure*) –

Returns void

getApplication(*$key*)
Returns a server application.

Parameters

- **$key** (*string*) – Name of application.

Returns Application The application object.

registerApplication(*$key*, *$application*)
Adds a new application object to the application storage.

Parameters

- **$key** (*string*) – Name of application.
- **$application** (*object*) – The application object

Returns void

configureProtocol()
Configures the protocol option

Wrench\Client

class Client
Client class

Represents a Wrench client

constant MAX_HANDSHAKE_RESPONSE

property uri
protected

property origin
protected

property socket
protected

property headers
protected array

Request headers

property protocol
protected Protocol

Protocol instance

property options
protected array

Options

property connected
protected boolean

Whether the client is connected

__construct (*$uri*, *$origin*, *$options = array()*)
Constructor

Parameters

- **$uri** (*string*) –
- **$origin** (*string*) – The origin to include in the handshake (required in later versions of the protocol)
- **$options** –

configure (*$options*)
Configure options

Parameters

- **$options** (*array*) –

Returns void

__destruct ()
Destructor

addRequestHeader (*$name*, *$value*)
Adds a request header to be included in the initial handshake

For example, to include a Cookie header

Parameters

- **$name** (*string*) –
- **$value** (*string*) –

Returns void

sendData (*$data*, *$type = 'text'*, *$masked = true*)
Sends data to the socket

Parameters

- **$data** (*string*) –
- **$type** (*string*) – Payload type
- **$masked** (*boolean*) –

Returns int bytes written

connect ()
Connect to the Wrench server

Returns boolean Whether a new connection was made

isConnected()
Whether the client is currently connected

Returns boolean

disconnect()

Wrench\Connection

class Connection
Represents a client connection on the server side

i.e. the *Server* manages a bunch of ‘Connection‘s

property manager
protected WrenchConnectionManager

The connection manager

property socket
protected Socket

Socket object

Wraps the client connection resource

property handshaked
protected boolean

Whether the connection has successfully handshaken

property application
protected Application

The application this connection belongs to

property ip
protected string

The IP address of the client

property port
protected int

The port of the client

property headers
protected array

The array of headers included with the original request (like Cookie for example). The headers specific to the web sockets handshaking have been stripped out.

property queryParams
protected array

The array of query parameters included in the original request. The array is in the format ‘key’ => ‘value’.

property id
protected string|null

Connection ID

property payload
protected

The current payload

property options
protected array

property protocol
protected Protocol

__construct(*ConnectionManager $manager*, *ServerClientSocket $socket*, *$options = array()*)
Constructor

Parameters

- **$manager** (ConnectionManager) –
- **$socket** (ServerClientSocket) –
- **$options** (*array*) –

getConnectionManager()
Gets the connection manager of this connection

Returns WrenchConnectionManager

configure(*$options*)

Parameters

- **$options** –

configureClientInformation()

configureClientId()
Configures the client ID

We hash the client ID to prevent leakage of information if another client happens to get a hold of an ID. The secret *must* be lengthy, and must be kept secret for this to work: otherwise it's trivial to search the space of possible IP addresses/ports (well, if not trivial, at least very fast).

onData(*$data*)
Data receiver

Called by the connection manager when the connection has received data

Parameters

- **$data** (*string*) –

handshake(*$data*)
Performs a websocket handshake

Parameters

- **$data** (*string*) –

export(*$data*)
Returns a string export of the given binary data

Parameters

- **$data** (*string*) –

Returns string

handle(*$data*)
Handle data received from the client

The data passed in may belong to several different frames across one or more protocols. It may not even contain a single complete frame. This method manages slotting the data into separate payload objects.

Parameters

- **$data** (*string*) –

handlePayload(*Payload $payload*)
Handle a complete payload received from the client

Parameters

- **$payload** (Payload) –

send(*$data*, *$type = Protocol::TYPE_TEXT*)
Sends the payload to the connection

Parameters

- **$data** –
- **$type** (*string*) –

Returns boolean

process()
Processes data on the socket

close(*$code = Protocol::CLOSE_NORMAL*)
Closes the connection according to the WebSocket protocol

Parameters

- **$code** –

Returns boolean

log(*$message*, *$priority = 'info'*)
Logs a message

Parameters

- **$message** (*string*) –
- **$priority** (*string*) –

getIp()
Gets the IP address of the connection

Returns string Usually dotted quad notation

getPort()
Gets the port of the connection

Returns int

getHeaders()
Gets the non-web-sockets headers included with the original request

Returns array

getQueryParams()
Gets the query parameters included with the original request

Returns array

getId()
Gets the connection ID

Returns string

getSocket()
Gets the socket object

Returns SocketServerClientSocket

getClientApplication()
Gets the client application

Returns Application

configureProtocol()
Configures the protocol option

Wrench\ConnectionManager

class ConnectionManager

property server
protected Server

property socket
protected Socket

Master socket

property connections
protected array<int

An array of client connections

property resources
protected array<int

An array of raw socket resources, corresponding to connections, roughly

property options
protected array

property protocol
protected Protocol

__construct(*Server $server*, *$options = array()*)
Constructor

Parameters

- **$server** (Server) –
- **$options** (*array*) –

count()

configure(*$options*)

Parameters

- **$options** –

getApplicationForPath(*$path*)
Gets the application associated with the given path

Parameters

- **$path** (*string*) –

configureMasterSocket()
Configures the main server socket

listen()
Listens on the main socket

Returns void

getAllResources()
Gets all resources

Returns array<int => resource)

getConnectionForClientSocket(*$socket*)
Returns the Connection associated with the specified socket resource

Parameters

- **$socket** (*resource*) –

Returns Connection

selectAndProcess()
Select and process an array of resources

processMasterSocket()
Process events on the master socket ($this->socket)

Returns void

createConnection(*$resource*)
Creates a connection from a socket resource

The create connection object is based on the options passed into the constructor ('connection_class', 'connection_options'). This connection instance and its associated socket resource are then stored in the manager.

Parameters

- **$resource** (*resource*) – A socket resource

Returns Connection

processClientSocket(*$socket*)
Process events on a client socket

Parameters

- **$socket** (*resource*) –

resourceId(*$resource*)
This server makes an explicit assumption: PHP resource types may be cast to a integer. Furthermore, we assume this is bijective. Both seem to be true in most circumstances, but may not be guaranteed.

This method (and $this->getResourceId()) exist to make this assumption explicit.

This is needed on the connection manager as well as on resources

Parameters

- **$resource** (*resource*) –

getUri()
Gets the connection manager's listening URI

Returns string

log(*$message*, *$priority = 'info'*)
Logs a message

Parameters

- **$message** (*string*) –
- **$priority** (*string*) –

getServer()

Returns WrenchServer

removeConnection(*Connection $connection*)
Removes a connection

Parameters

- **$connection** (Connection) –

configureProtocol()
Configures the protocol option

Wrench\Exception

Wrench\Exception\BadRequestException

class BadRequestException

property message
protected

property code
protected

property file
protected

property line
protected

__construct(*$message = null*, *$code = null*, *$previous = null*)

Parameters

- **$message** –
- **$code** –
- **$previous** (Exception) –

__clone()

getMessage()

getCode()

getFile()

getLine()

getTrace()

getPrevious()

getTraceAsString()

__toString()

Wrench\Exception\CloseException

class CloseException
Close connection exception

property message
protected

property code
protected

property file
protected

property line
protected

__construct(*$message = null*, *$code = null*, *$previous = null*)

Parameters

- **$message** –
- **$code** –
- **$previous** (Exception) –

__clone()

getMessage()

getCode()

getFile()

getLine()

getTrace()

getPrevious()

getTraceAsString()

__toString()

Wrench\Exception\ConnectionException

class ConnectionException

property message
protected

property code
protected

property file
protected

property line
protected

__clone()

__construct(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

getMessage()

getCode()

getFile()

getLine()

getTrace()

getPrevious()

getTraceAsString()

__toString()

Wrench\Exception\Exception

class Exception

property message
protected

property code
protected

property file
protected

property line
protected

__clone()

__construct(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

getMessage()

getCode()

getFile()

getLine()

getTrace()

getPrevious()

getTraceAsString()

__toString()

Wrench\Exception\FrameException

class FrameException

property message
protected

property code
protected

property file
protected

property line
protected

__clone()

__construct(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

getMessage()

getCode()

getFile()

getLine()

getTrace()

getPrevious()

getTraceAsString()

__toString()

Wrench\Exception\HandshakeException

class HandshakeException

property `message`
protected

property `code`
protected

property `file`
protected

property `line`
protected

`__construct` (*$message = null*, *$code = null*, *$previous = null*)

Parameters

- **`$message`** –
- **`$code`** –
- **`$previous`** (`Exception`) –

`__clone`()

`getMessage`()

`getCode`()

`getFile`()

`getLine`()

`getTrace`()

`getPrevious`()

`getTraceAsString`()

`__toString`()

Wrench\Exception\InvalidOriginException

class `InvalidOriginException`
Invalid origin exception

property `message`
protected

property `code`
protected

property `file`
protected

property `line`
protected

`__construct` (*$message = null*, *$code = null*, *$previous = null*)

Parameters

- **`$message`** –
- **`$code`** –
- **`$previous`** (`Exception`) –

__clone()

getMessage()

getCode()

getFile()

getLine()

getTrace()

getPrevious()

getTraceAsString()

__toString()

Wrench\Exception\PayloadException

class PayloadException

property message
protected

property code
protected

property file
protected

property line
protected

__clone()

__construct(*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

getMessage()

getCode()

getFile()

getLine()

getTrace()

getPrevious()

getTraceAsString()

__toString()

Wrench\Exception\RateLimiterException

class `RateLimiterException`

property `message`
protected

property `code`
protected

property `file`
protected

property `line`
protected

`__construct` (*$message = null*, *$code = null*, *$previous = null*)

Parameters

- **`$message`** –
- **`$code`** –
- **`$previous`** (`Exception`) –

`__clone` ()

`getMessage` ()

`getCode` ()

`getFile` ()

`getLine` ()

`getTrace` ()

`getPrevious` ()

`getTraceAsString` ()

`__toString` ()

Wrench\Exception\SocketException

class `SocketException`

property `message`
protected

property `code`
protected

property `file`
protected

property `line`
protected

`__clone` ()

`__construct` (*$message*, *$code*, *$previous*)

Parameters

- **$message** –
- **$code** –
- **$previous** –

getMessage()

getCode()

getFile()

getLine()

getTrace()

getPrevious()

getTraceAsString()

__toString()

Wrench\Frame

Wrench\Frame\Frame

class Frame

Represents a WebSocket frame

property length

protected int

The frame data length

property type

protected int

The type of this payload

property buffer

protected string

The buffer

May not be a complete payload, because this frame may still be receiving data. See

property payload

protected string

The enclosed frame payload

May not be a complete payload, because this frame might indicate a continuation frame. See isFinal() versus isComplete()

getLength()

Gets the length of the payload

Returns int

encode(*$data*, *$type = Protocol::TYPE_TEXT*, *$masked = false*)

Resets the frame and encodes the given data into it

Parameters

- **$data** (*string*) –
- **$type** (*int*) –
- **$masked** (*boolean*) –

Returns Frame

isFinal()
Whether the frame is the final one in a continuation

Returns boolean

getType()

Returns int

decodeFramePayloadFromBuffer()
Decodes a frame payload from the buffer

Returns void

getExpectedBufferLength()
Gets the expected length of the buffer once all the data has been receieved

Returns int

isComplete()
Whether the frame is complete

Returns boolean

receiveData(*$data*)
Receieves data into the frame

Parameters

- **$data** –

getRemainingData()
Gets the remaining number of bytes before this frame will be complete

Returns number

isWaitingForData()
Whether this frame is waiting for more data

Returns boolean

getFramePayload()
Gets the contents of the frame payload

The frame must be complete to call this method.

Returns string

getFrameBuffer()
Gets the contents of the frame buffer

This is the encoded value, receieved into the frame with recieveData().

Returns string binary

getBufferLength()
Gets the expected length of the frame payload

Returns int

Wrench\Frame\HybiFrame

class `HybiFrame`

property `masked`
protected boolean

Whether the payload is masked

property `mask`
protected string

Masking key

property `offset_payload`
protected int

Byte offsets

property `offset_mask`
protected

property `length`
protected int

The frame data length

property `type`
protected int

The type of this payload

property `buffer`
protected string

The buffer

May not be a complete payload, because this frame may still be receiving data. See

property `payload`
protected string

The enclosed frame payload

May not be a complete payload, because this frame might indicate a continuation frame. See isFinal() versus isComplete()

`encode` (*$payload*, *$type = Protocol::TYPE_TEXT*, *$masked = false*)

Parameters

- **`$payload`** –
- **`$type`** –
- **`$masked`** –

`mask` (*$payload*)
Masks/Unmasks the frame

Parameters

- **`$payload`** (*string*) –

Returns string

unmask(*$payload*)
Masks a payload

Parameters

- **$payload**(*string*) –

Returns string

receiveData(*$data*)

Parameters

- **$data** –

getMask()
Gets the mask

Returns string

generateMask()
Generates a suitable masking key

Returns string

isMasked()
Whether the frame is masked

Returns boolean

getExpectedBufferLength()

getPayloadOffset()
Gets the offset of the payload in the frame

Returns int

getMaskOffset()
Gets the offset in the frame to the masking bytes

Returns int

getLength()

getInitialLength()
Gets the inital length value, stored in the first length byte

This determines how the rest of the length value is parsed out of the frame.

Returns int

getLengthSize()
Returns the byte size of the length part of the frame

Not including the initial 7 bit part

Returns int

getMaskSize()
Returns the byte size of the mask part of the frame

Returns int

decodeFramePayloadFromBuffer()

isFinal()

getType()

isComplete()
Whether the frame is complete

Returns boolean

getRemainingData()
Gets the remaining number of bytes before this frame will be complete

Returns number

isWaitingForData()
Whether this frame is waiting for more data

Returns boolean

getFramePayload()
Gets the contents of the frame payload

The frame must be complete to call this method.

Returns string

getFrameBuffer()
Gets the contents of the frame buffer

This is the encoded value, receieved into the frame with recieveData().

Returns string binary

getBufferLength()
Gets the expected length of the frame payload

Returns int

Wrench\Listener

Wrench\Listener\HandshakeRequestListener

class HandshakeRequestListener

onHandshakeRequest(*Connection $connection*, *$path*, *$origin*, *$key*, *$extensions*)
Handshake request listener

Parameters

- **$connection** (Connection) –
- **$path** (*string*) –
- **$origin** (*string*) –
- **$key** (*string*) –
- **$extensions** (*array*) –

Wrench\Listener\Listener

class `Listener`

`listen` (*Server $server*)

Parameters

- **`$server`** (`Server`) –

Wrench\Listener\OriginPolicy

class `OriginPolicy`

property `allowed`
protected

`__construct` (*$allowed*)

Parameters

- **`$allowed`** –

`onHandshakeRequest` (*Connection $connection*, *$path*, *$origin*, *$key*, *$extensions*)
Handshake request listener

Closes the connection on handshake from an origin that isn't allowed

Parameters

- **`$connection`** (`Connection`) –
- **`$path`** (*string*) –
- **`$origin`** (*string*) –
- **`$key`** (*string*) –
- **`$extensions`** (*array*) –

`isAllowed` (*$origin*)
Whether the specified origin is allowed under this policy

Parameters

- **`$origin`** (*string*) –

Returns boolean

`listen` (*Server $server*)

Parameters

- **`$server`** (`Server`) –

Wrench\Listener\RateLimiter

class `RateLimiter`

property server
protected Server

The server being limited

property ips
protected array<int>

Connection counts per IP address

property requests
protected array<array<int>>

Request tokens per IP address

property options
protected array

property protocol
protected Protocol

__construct (*$options = array()*)
Constructor

Parameters

- **$options** (*array*) –

configure (*$options*)

Parameters

- **$options** (*array*) –

listen (*Server $server*)

Parameters

- **$server** (Server) –

onSocketConnect (*$socket*, *$connection*)
Event listener

Parameters

- **$socket** (*resource*) –
- **$connection** (Connection) –

onSocketDisconnect (*$socket*, *$connection*)
Event listener

Parameters

- **$socket** (*resource*) –
- **$connection** (Connection) –

onClientData (*$socket*, *$connection*)
Event listener

Parameters

- **$socket** (*resource*) –
- **$connection** (Connection) –

checkConnections(*$connection*)
Idempotent

Parameters

- **$connection** (Connection) –

checkConnectionsPerIp(*$connection*)
NOT idempotent, call once per connection

Parameters

- **$connection** (Connection) –

releaseConnection(*$connection*)
NOT idempotent, call once per disconnection

Parameters

- **$connection** (Connection) –

checkRequestsPerMinute(*$connection*)
NOT idempotent, call once per data

Parameters

- **$connection** (Connection) –

limit(*$connection*, *$limit*)
Limits the given connection

Parameters

- **$connection** (Connection) –
- **$limit** (*string*) – Reason

log(*$message*, *$priority = 'info'*)
Logger

Parameters

- **$message** (*string*) –
- **$priority** (*string*) –

configureProtocol()
Configures the protocol option

Wrench\Payload

Wrench\Payload\HybiPayload

class HybiPayload
Gets a HyBi payload

property frames
protected array<Frame>

A payload may consist of one or more frames

getFrame()

getCurrentFrame()
Gets the current frame for the payload

Returns mixed

getReceivingFrame()
Gets the frame into which data should be receieved

Returns Frame

isComplete()
Whether the payload is complete

Returns boolean

encode(*$data*, *$type = Protocol::TYPE_TEXT*, *$masked = false*)
Encodes a payload

Parameters

- **$data** (*string*) –
- **$type** (*int*) –
- **$masked** (*boolean*) –

Returns Payload

getRemainingData()
Gets the number of remaining bytes before this payload will be complete

May return 0 (no more bytes required) or null (unknown number of bytes required).

Returns number|NULL

isWaitingForData()
Whether this payload is waiting for more data

Returns boolean

sendToSocket(*Socket $socket*)

Parameters

- **$socket** (Socket) –

Returns boolean

receiveData(*$data*)
Receive raw data into the payload

Parameters

- **$data** (*string*) –

Returns void

getPayload()

Returns string

__toString()

Returns string

getType()
Gets the type of the payload

The type of a payload is taken from its first frame

Returns int

Wrench\Payload\Payload

class `Payload`
Payload class

Represents a WebSocket protocol payload, which may be made up of multiple frames.

property `frames`
protected array<Frame>

A payload may consist of one or more frames

`getCurrentFrame`()
Gets the current frame for the payload

Returns mixed

`getReceivingFrame`()
Gets the frame into which data should be receieved

Returns Frame

`getFrame`()
Get a frame object

Returns Frame

`isComplete`()
Whether the payload is complete

Returns boolean

`encode`(*$data*, *$type = Protocol::TYPE_TEXT*, *$masked = false*)
Encodes a payload

Parameters

- **`$data`** (*string*) –
- **`$type`** (*int*) –
- **`$masked`** (*boolean*) –

Returns Payload

`getRemainingData`()
Gets the number of remaining bytes before this payload will be complete

May return 0 (no more bytes required) or null (unknown number of bytes required).

Returns number|NULL

`isWaitingForData`()
Whether this payload is waiting for more data

Returns boolean

`sendToSocket`(*Socket $socket*)

Parameters

- **`$socket`** (Socket) –

Returns boolean

receiveData(*$data*)
Receive raw data into the payload

Parameters

- **$data** (*string*) –

Returns void

getPayload()

Returns string

__toString()

Returns string

getType()
Gets the type of the payload

The type of a payload is taken from its first frame

Returns int

Wrench\Protocol

Wrench\Protocol\Hybi10Protocol

class Hybi10Protocol
http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-10

constant SCHEME_WEBSOCKET
Relevant schemes

constant HEADER_HOST
HTTP headers

constant HTTP_SWITCHING_PROTOCOLS
HTTP error statuses

constant CLOSE_NORMAL
Close statuses

constant TYPE_CONTINUATION
Frame types

%x0 denotes a continuation frame %x1 denotes a text frame %x2 denotes a binary frame %x3-7 are reserved for further non-control frames %x8 denotes a connection close %x9 denotes a ping %xA denotes a pong %xB-F are reserved for further control frames

constant MAGIC_GUID
Magic GUID

Used in the WebSocket accept header

constant UPGRADE_VALUE

The request MUST contain an |Upgrade| header field whose value MUST include the "websocket" keyword.

constant CONNECTION_VALUE

The request MUST contain a |Connection| header field whose value MUST include the "Upgrade" token.

constant `REQUEST_LINE_FORMAT`
Request line format

constant `REQUEST_LINE_REGEX`
Request line regex

Used for parsing requested path

constant `RESPONSE_LINE_FORMAT`
Response line format

constant `HEADER_LINE_FORMAT`
Header line format

property `schemes`
protected array<string>

Valid schemes

property `closeReasons`
array<int

Close status codes

property `frameTypes`
array<string

Frame types

property `httpResponses`
array<int

HTTP errors

`getVersion`()

`acceptsVersion`(*$version*)
This is our most recent protocol class

Parameters

- **`$version`** –

`getPayload`()

`generateKey`()
Generates a key suitable for use in the protocol

This base implementation returns a 16-byte (128 bit) random key as a binary string.

Returns string

`getRequestHandshake`(*$uri*, *$key*, *$origin*, *$headers = array()*)
Gets request handshake string

The leading line from the client follows the Request-Line format. The leading line from the server follows the Status-Line format. The Request-Line and Status-Line productions are defined in [RFC2616].

An unordered set of header fields comes after the leading line in both cases. The meaning of these header fields is specified in Section 4 of this document. Additional header fields may also be present, such as cookies [RFC6265]. The format and parsing of headers is as defined in [RFC2616].

Parameters

- **$uri** (*string*) – WebSocket URI, e.g. ws://example.org:8000/chat
- **$key** (*string*) – 16 byte binary string key
- **$origin** (*string*) – Origin of the request
- **$headers** –

Returns string

getResponseHandshake(*$key*, *$headers = array()*)
Gets a handshake response body

Parameters

- **$key** (*string*) –
- **$headers** (*array*) –

getResponseError(*$e*, *$headers = array()*)
Gets a response to an error in the handshake

Parameters

- **$e** (*int|Exception*) – Exception or HTTP error
- **$headers** (*array*) –

getHttpResponse(*$status*, *$headers = array()*)
Gets an HTTP response

Parameters

- **$status** (*int*) –
- **$headers** (*array*) –

validateResponseHandshake(*$response*, *$key*)

Parameters

- **$response** (*unknown_type*) –
- **$key** (*unknown_type*) –

Returns boolean

getEncodedHash(*$key*)
Gets an encoded hash for a key

Parameters

- **$key** (*string*) –

Returns string

validateRequestHandshake(*$request*)
Validates a request handshake

Parameters

- **$request** (*string*) –

getCloseFrame(*$e*)
Gets a suitable WebSocket close frame

Parameters

- **$e** (*Exception|int*) –

validateUri (*$uri*)
Validates a WebSocket URI

Parameters

- **$uri** (*string*) –

Returns array(string $scheme, string $host, int $port, string $path)

validateSocketUri (*$uri*)
Validates a socket URI

Parameters

- **$uri** (*string*) –

Returns array(string $scheme, string $host, string $port)

validateOriginUri (*$origin*)
Validates an origin URI

Parameters

- **$origin** (*string*) –

Returns string

validateRequestLine (*$line*)
Validates a request line

Parameters

- **$line** (*string*) –

getAcceptValue (*$encoded_key*)
Gets the expected accept value for a handshake response

Note that the protocol calls for the base64 encoded value to be hashed, not the original 16 byte random key.

Parameters

- **$encoded_key** –

getHeaders (*$response*, *$request_line = null*)
Gets the headers from a full response

Parameters

- **$response** (*string*) –
- **$request_line** –

Returns array()

getRequestHeaders (*$response*)
Gets request headers

Parameters

- **$response** (*string*) –

Returns array<string, array<string>> The request line, and an array of headers

validateScheme (*$scheme*)
Validates a scheme

Parameters

- **$scheme** (*string*) –

Returns string Underlying scheme

getDefaultRequestHeaders (*$host*, *$key*, *$origin*)
Gets the default request headers

Parameters

- **$host** (*string*) –
- **$key** (*string*) –
- **$origin** (*string*) –

Returns multitype:unknown string NULL

getSuccessResponseHeaders (*$key*)
Gets the default response headers

Parameters

- **$key** (*string*) –

getPort (*$scheme*)
Gets the default port for a scheme

By default, the WebSocket Protocol uses port 80 for regular WebSocket connections and port 443 for WebSocket connections tunneled over Transport Layer Security

Parameters

- **$scheme** –

Returns int

Wrench\Protocol\HybiProtocol

class HybiProtocol

constant SCHEME_WEBSOCKET
Relevant schemes

constant HEADER_HOST
HTTP headers

constant HTTP_SWITCHING_PROTOCOLS
HTTP error statuses

constant CLOSE_NORMAL
Close statuses

constant TYPE_CONTINUATION
Frame types

%x0 denotes a continuation frame %x1 denotes a text frame %x2 denotes a binary frame %x3-7 are reserved for further non-control frames %x8 denotes a connection close %x9 denotes a ping %xA denotes a pong %xB-F are reserved for further control frames

constant MAGIC_GUID
Magic GUID

Used in the WebSocket accept header

constant `UPGRADE_VALUE`

The request MUST contain an |Upgrade| header field whose value MUST include the "websocket" keyword.

constant `CONNECTION_VALUE`

The request MUST contain a |Connection| header field whose value MUST include the "Upgrade" token.

constant `REQUEST_LINE_FORMAT`
Request line format

constant `REQUEST_LINE_REGEX`
Request line regex

Used for parsing requested path

constant `RESPONSE_LINE_FORMAT`
Response line format

constant `HEADER_LINE_FORMAT`
Header line format

property `schemes`
protected array<string>

Valid schemes

property `closeReasons`
array<int

Close status codes

property `frameTypes`
array<string

Frame types

property `httpResponses`
array<int

HTTP errors

`getPayload`()

`getVersion`()
Gets a version number

`acceptsVersion`(*$version*)
Subclasses should implement this method and return a boolean to the given version string, as to whether they would like to accept requests from user agents that specify that version.

Parameters

- **`$version`** –

Returns boolean

`generateKey`()
Generates a key suitable for use in the protocol

This base implementation returns a 16-byte (128 bit) random key as a binary string.

Returns string

getRequestHandshake(*$uri*, *$key*, *$origin*, *$headers = array()*)
Gets request handshake string

The leading line from the client follows the Request-Line format. The leading line from the server follows the Status-Line format. The Request-Line and Status-Line productions are defined in [RFC2616].

An unordered set of header fields comes after the leading line in both cases. The meaning of these header fields is specified in Section 4 of this document. Additional header fields may also be present, such as cookies [RFC6265]. The format and parsing of headers is as defined in [RFC2616].

Parameters

- **$uri** (*string*) – WebSocket URI, e.g. ws://example.org:8000/chat
- **$key** (*string*) – 16 byte binary string key
- **$origin** (*string*) – Origin of the request
- **$headers** –

Returns string

getResponseHandshake(*$key*, *$headers = array()*)
Gets a handshake response body

Parameters

- **$key** (*string*) –
- **$headers** (*array*) –

getResponseError(*$e*, *$headers = array()*)
Gets a response to an error in the handshake

Parameters

- **$e** (*int|Exception*) – Exception or HTTP error
- **$headers** (*array*) –

getHttpResponse(*$status*, *$headers = array()*)
Gets an HTTP response

Parameters

- **$status** (*int*) –
- **$headers** (*array*) –

validateResponseHandshake(*$response*, *$key*)

Parameters

- **$response** (*unknown_type*) –
- **$key** (*unknown_type*) –

Returns boolean

getEncodedHash(*$key*)
Gets an encoded hash for a key

Parameters

- **$key** (*string*) –

Returns string

validateRequestHandshake(*$request*)
Validates a request handshake

Parameters

- **$request** (*string*) –

getCloseFrame(*$e*)
Gets a suitable WebSocket close frame

Parameters

- **$e** (*Exception|int*) –

validateUri(*$uri*)
Validates a WebSocket URI

Parameters

- **$uri** (*string*) –

Returns array(string $scheme, string $host, int $port, string $path)

validateSocketUri(*$uri*)
Validates a socket URI

Parameters

- **$uri** (*string*) –

Returns array(string $scheme, string $host, string $port)

validateOriginUri(*$origin*)
Validates an origin URI

Parameters

- **$origin** (*string*) –

Returns string

validateRequestLine(*$line*)
Validates a request line

Parameters

- **$line** (*string*) –

getAcceptValue(*$encoded_key*)
Gets the expected accept value for a handshake response

Note that the protocol calls for the base64 encoded value to be hashed, not the original 16 byte random key.

Parameters

- **$encoded_key** –

getHeaders(*$response*, *$request_line = null*)
Gets the headers from a full response

Parameters

- **$response** (*string*) –
- **$request_line** –

Returns array()

getRequestHeaders (*$response*)
Gets request headers

Parameters

- **$response** (*string*) –

Returns array<string, array<string>> The request line, and an array of headers

validateScheme (*$scheme*)
Validates a scheme

Parameters

- **$scheme** (*string*) –

Returns string Underlying scheme

getDefaultRequestHeaders (*$host*, *$key*, *$origin*)
Gets the default request headers

Parameters

- **$host** (*string*) –
- **$key** (*string*) –
- **$origin** (*string*) –

Returns multitype:unknown string NULL

getSuccessResponseHeaders (*$key*)
Gets the default response headers

Parameters

- **$key** (*string*) –

getPort (*$scheme*)
Gets the default port for a scheme

By default, the WebSocket Protocol uses port 80 for regular WebSocket connections and port 443 for WebSocket connections tunneled over Transport Layer Security

Parameters

- **$scheme** –

Returns int

Wrench\Protocol\Protocol

class Protocol
Definitions and implementation helpers for the Wrenchs protocol

Based on RFC 6455: http://tools.ietf.org/html/rfc6455

constant SCHEME_WEBSOCKET
Relevant schemes

constant HEADER_HOST
HTTP headers

constant HTTP_SWITCHING_PROTOCOLS
HTTP error statuses

constant `CLOSE_NORMAL`
Close statuses

constant `TYPE_CONTINUATION`
Frame types

%x0 denotes a continuation frame %x1 denotes a text frame %x2 denotes a binary frame %x3-7 are reserved for further non-control frames %x8 denotes a connection close %x9 denotes a ping %xA denotes a pong %xB-F are reserved for further control frames

constant `MAGIC_GUID`
Magic GUID

Used in the WebSocket accept header

constant `UPGRADE_VALUE`

The request MUST contain an |Upgrade| header field whose value MUST include the "websocket" keyword.

constant `CONNECTION_VALUE`

The request MUST contain a |Connection| header field whose value MUST include the "Upgrade" token.

constant `REQUEST_LINE_FORMAT`
Request line format

constant `REQUEST_LINE_REGEX`
Request line regex

Used for parsing requested path

constant `RESPONSE_LINE_FORMAT`
Response line format

constant `HEADER_LINE_FORMAT`
Header line format

property `schemes`
protected array<string>

Valid schemes

property `closeReasons`
array<int

Close status codes

property `frameTypes`
array<string

Frame types

property `httpResponses`
array<int

HTTP errors

`getVersion`()
Gets a version number

`acceptsVersion`(*$version*)
Subclasses should implement this method and return a boolean to the given version string, as to whether they would like to accept requests from user agents that specify that version.

Parameters

- **$version** –

Returns boolean

getPayload()
Gets a payload instance, suitable for use in decoding/encoding protocol frames

Returns Payload

generateKey()
Generates a key suitable for use in the protocol

This base implementation returns a 16-byte (128 bit) random key as a binary string.

Returns string

getRequestHandshake(*$uri*, *$key*, *$origin*, *$headers = array()*)
Gets request handshake string

The leading line from the client follows the Request-Line format. The leading line from the server follows the Status-Line format. The Request-Line and Status-Line productions are defined in [RFC2616].

An unordered set of header fields comes after the leading line in both cases. The meaning of these header fields is specified in Section 4 of this document. Additional header fields may also be present, such as cookies [RFC6265]. The format and parsing of headers is as defined in [RFC2616].

Parameters

- **$uri** (*string*) – WebSocket URI, e.g. ws://example.org:8000/chat
- **$key** (*string*) – 16 byte binary string key
- **$origin** (*string*) – Origin of the request
- **$headers** –

Returns string

getResponseHandshake(*$key*, *$headers = array()*)
Gets a handshake response body

Parameters

- **$key** (*string*) –
- **$headers** (*array*) –

getResponseError(*$e*, *$headers = array()*)
Gets a response to an error in the handshake

Parameters

- **$e** (*int|Exception*) – Exception or HTTP error
- **$headers** (*array*) –

getHttpResponse(*$status*, *$headers = array()*)
Gets an HTTP response

Parameters

- **$status** (*int*) –
- **$headers** (*array*) –

validateResponseHandshake(*$response*, *$key*)

Parameters

- **$response** (*unknown_type*) –
- **$key** (*unknown_type*) –

Returns boolean

getEncodedHash(*$key*)
Gets an encoded hash for a key

Parameters

- **$key** (*string*) –

Returns string

validateRequestHandshake(*$request*)
Validates a request handshake

Parameters

- **$request** (*string*) –

getCloseFrame(*$e*)
Gets a suitable WebSocket close frame

Parameters

- **$e** (*Exception|int*) –

validateUri(*$uri*)
Validates a WebSocket URI

Parameters

- **$uri** (*string*) –

Returns array(string $scheme, string $host, int $port, string $path)

validateSocketUri(*$uri*)
Validates a socket URI

Parameters

- **$uri** (*string*) –

Returns array(string $scheme, string $host, string $port)

validateOriginUri(*$origin*)
Validates an origin URI

Parameters

- **$origin** (*string*) –

Returns string

validateRequestLine(*$line*)
Validates a request line

Parameters

- **$line** (*string*) –

getAcceptValue(*$encoded_key*)
Gets the expected accept value for a handshake response

Note that the protocol calls for the base64 encoded value to be hashed, not the original 16 byte random key.

Parameters

- **$encoded_key** –

getHeaders(*$response*, *$request_line = null*)
Gets the headers from a full response

Parameters

- **$response** (*string*) –
- **$request_line** –

Returns array()

getRequestHeaders(*$response*)
Gets request headers

Parameters

- **$response** (*string*) –

Returns array<string, array<string>> The request line, and an array of headers

validateScheme(*$scheme*)
Validates a scheme

Parameters

- **$scheme** (*string*) –

Returns string Underlying scheme

getDefaultRequestHeaders(*$host*, *$key*, *$origin*)
Gets the default request headers

Parameters

- **$host** (*string*) –
- **$key** (*string*) –
- **$origin** (*string*) –

Returns multitype:unknown string NULL

getSuccessResponseHeaders(*$key*)
Gets the default response headers

Parameters

- **$key** (*string*) –

getPort(*$scheme*)
Gets the default port for a scheme

By default, the WebSocket Protocol uses port 80 for regular WebSocket connections and port 443 for WebSocket connections tunneled over Transport Layer Security

Parameters

- **$scheme** –

Returns int

Wrench\Protocol\Rfc6455Protocol

class `Rfc6455Protocol`
This is the version of websockets used by Chrome versions 17 through 19.

constant `SCHEME_WEBSOCKET`
Relevant schemes

constant `HEADER_HOST`
HTTP headers

constant `HTTP_SWITCHING_PROTOCOLS`
HTTP error statuses

constant `CLOSE_NORMAL`
Close statuses

constant `TYPE_CONTINUATION`
Frame types

%x0 denotes a continuation frame %x1 denotes a text frame %x2 denotes a binary frame %x3-7 are reserved for further non-control frames %x8 denotes a connection close %x9 denotes a ping %xA denotes a pong %xB-F are reserved for further control frames

constant `MAGIC_GUID`
Magic GUID

Used in the WebSocket accept header

constant `UPGRADE_VALUE`

The request MUST contain an |Upgrade| header field whose value MUST include the "websocket" keyword.

constant `CONNECTION_VALUE`

The request MUST contain a |Connection| header field whose value MUST include the "Upgrade" token.

constant `REQUEST_LINE_FORMAT`
Request line format

constant `REQUEST_LINE_REGEX`
Request line regex

Used for parsing requested path

constant `RESPONSE_LINE_FORMAT`
Response line format

constant `HEADER_LINE_FORMAT`
Header line format

property `schemes`
protected array<string>

Valid schemes

property `closeReasons`
array<int

Close status codes

property `frameTypes`
array<string

Frame types

property `httpResponses`
array<int

HTTP errors

`getVersion`()

`acceptsVersion`(*$version*)
This is our most recent protocol class

Parameters

- **`$version`** –

`getPayload`()

`generateKey`()
Generates a key suitable for use in the protocol

This base implementation returns a 16-byte (128 bit) random key as a binary string.

Returns string

`getRequestHandshake`(*$uri*, *$key*, *$origin*, *$headers = array()*)
Gets request handshake string

The leading line from the client follows the Request-Line format. The leading line from the server follows the Status-Line format. The Request-Line and Status-Line productions are defined in [RFC2616].

An unordered set of header fields comes after the leading line in both cases. The meaning of these header fields is specified in Section 4 of this document. Additional header fields may also be present, such as cookies [RFC6265]. The format and parsing of headers is as defined in [RFC2616].

Parameters

- **`$uri`** (*`string`*) – WebSocket URI, e.g. ws://example.org:8000/chat
- **`$key`** (*`string`*) – 16 byte binary string key
- **`$origin`** (*`string`*) – Origin of the request
- **`$headers`** –

Returns string

`getResponseHandshake`(*$key*, *$headers = array()*)
Gets a handshake response body

Parameters

- **`$key`** (*`string`*) –
- **`$headers`** (*`array`*) –

`getResponseError`(*$e*, *$headers = array()*)
Gets a response to an error in the handshake

Parameters

- **`$e`** (*`int|Exception`*) – Exception or HTTP error
- **`$headers`** (*`array`*) –

getHttpResponse(*$status*, *$headers = array()*)
Gets an HTTP response

Parameters

- **$status** (*int*) –
- **$headers** (*array*) –

validateResponseHandshake(*$response*, *$key*)

Parameters

- **$response** (*unknown_type*) –
- **$key** (*unknown_type*) –

Returns boolean

getEncodedHash(*$key*)
Gets an encoded hash for a key

Parameters

- **$key** (*string*) –

Returns string

validateRequestHandshake(*$request*)
Validates a request handshake

Parameters

- **$request** (*string*) –

getCloseFrame(*$e*)
Gets a suitable WebSocket close frame

Parameters

- **$e** (*Exception|int*) –

validateUri(*$uri*)
Validates a WebSocket URI

Parameters

- **$uri** (*string*) –

Returns array(string $scheme, string $host, int $port, string $path)

validateSocketUri(*$uri*)
Validates a socket URI

Parameters

- **$uri** (*string*) –

Returns array(string $scheme, string $host, string $port)

validateOriginUri(*$origin*)
Validates an origin URI

Parameters

- **$origin** (*string*) –

Returns string

validateRequestLine(*$line*)
Validates a request line

Parameters

- **$line** (*string*) –

getAcceptValue(*$encoded_key*)
Gets the expected accept value for a handshake response

Note that the protocol calls for the base64 encoded value to be hashed, not the original 16 byte random key.

Parameters

- **$encoded_key** –

getHeaders(*$response*, *$request_line = null*)
Gets the headers from a full response

Parameters

- **$response** (*string*) –
- **$request_line** –

Returns array()

getRequestHeaders(*$response*)
Gets request headers

Parameters

- **$response** (*string*) –

Returns array<string, array<string>> The request line, and an array of headers

validateScheme(*$scheme*)
Validates a scheme

Parameters

- **$scheme** (*string*) –

Returns string Underlying scheme

getDefaultRequestHeaders(*$host*, *$key*, *$origin*)
Gets the default request headers

Parameters

- **$host** (*string*) –
- **$key** (*string*) –
- **$origin** (*string*) –

Returns multitype:unknown string NULL

getSuccessResponseHeaders(*$key*)
Gets the default response headers

Parameters

- **$key** (*string*) –

getPort(*$scheme*)
Gets the default port for a scheme

By default, the WebSocket Protocol uses port 80 for regular WebSocket connections and port 443 for WebSocket connections tunneled over Transport Layer Security

Parameters

- **$scheme** –

Returns int

Wrench\Resource

class Resource
Resource interface

getResourceId()

getResource()

Wrench\Server

class Server
WebSocket server

The server extends socket, which provides the master socket resource. This resource is listened to, and an array of clients managed.

constant EVENT_SOCKET_CONNECT
Events

property uri
protected string

The URI of the server

property options
protected array

Options

property logger
protected Closure

A logging callback

The default callback simply prints to stdout. You can pass your own logger in the options array. It should take a string message and string priority as parameters.

property listeners
protected array<string

Event listeners

Add listeners using the addListener() method.

property connectionManager
protected ConnectionManager

Connection manager

property applications
protected array<string

Applications

property protocol
protected Protocol

__construct (*$uri*, *$options = array()*)
Constructor

Parameters

- **$uri** (*string*) – Websocket URI, e.g. ws://localhost:8000/, path will be ignored
- **$options** (*array*) – (optional) See configure

configure (*$options*)
Configure options

Options include - socket_class => The socket class to use, defaults to ServerSocket - socket_options => An array of socket options - logger => Closure($message, $priority = 'info'), used for logging

Parameters

- **$options** (*array*) –

Returns void

configureLogger ()
Configures the logger

Returns void

configureConnectionManager ()
Configures the connection manager

Returns void

getConnectionManager ()
Gets the connection manager

Returns WrenchConnectionManager

getUri ()

Returns string

setLogger (*$logger*)
Sets a logger

Parameters

- **$logger** (*Closure*) –

Returns void

run ()
Main server loop

Returns void This method does not return!

log (*$message*, *$priority = 'info'*)
Logs a message to the server log

The default logger simply prints the message to stdout. You can provide a logging closure. This is useful, for instance, if you've daemonized and closed STDOUT.

Parameters

- **$message** (*string*) – Message to display.
- **$priority** –

Returns void

notify (*$event*, *$arguments = array()*)
Notifies listeners of an event

Parameters

- **$event** (*string*) –
- **$arguments** (*array*) – Event arguments

Returns void

addListener (*$event*, *$callback*)
Adds a listener

Provide an event (see the Server::EVENT_* constants) and a callback closure. Some arguments may be provided to your callback, such as the connection the caused the event.

Parameters

- **$event** (*string*) –
- **$callback** (*Closure*) –

Returns void

getApplication (*$key*)
Returns a server application.

Parameters

- **$key** (*string*) – Name of application.

Returns Application The application object.

registerApplication (*$key*, *$application*)
Adds a new application object to the application storage.

Parameters

- **$key** (*string*) – Name of application.
- **$application** (*object*) – The application object

Returns void

configureProtocol ()
Configures the protocol option

Wrench\Socket

Wrench\Socket\ClientSocket

class ClientSocket

Options:

- timeout_connect => int, seconds, default 2

constant `TIMEOUT_CONNECT`
Default connection timeout

constant `TIMEOUT_SOCKET`
Default timeout for socket operations (reads, writes)

constant `DEFAULT_RECEIVE_LENGTH`

constant `NAME_PART_IP`
Socket name parts

property `scheme`
protected

property `host`
protected

property `port`
protected

property `socket`
protected resource

property `context`
protected

Stream context

property `connected`
protected boolean

Whether the socket is connected to a server

Note, the connection may not be ready to use, but the socket is connected at least. See $handshaked, and other properties in subclasses.

property `firstRead`
protected boolean

Whether the current read is the first one to the socket

property `name`
protected string

The socket name according to stream_socket_get_name

property `options`
protected array

property `protocol`
protected Protocol

`configure`(*$options*)

Parameters

- **`$options`** –

`connect`()
Connects to the given socket

`reconnect`()

`getSocketStreamContextOptions`()

`getSslStreamContextOptions`()

__construct(*$uri*, *$options = array()*)
URI Socket constructor

Parameters

- **$uri** (*string*) – WebSocket URI, e.g. ws://example.org:8000/chat
- **$options** –

getUri()
Gets the canonical/normalized URI for this socket

Returns string

getName()

getHost()
Gets the host name

getPort()

getStreamContext(*$listen = false*)
Gets a stream context

Parameters

- **$listen** –

getNamePart(*$name*, *$part*)
Gets part of the name of the socket

PHP seems to return IPV6 address/port combos like this: ::1:1234, where ::1 is the address and 1234 the port So, the part number here is either the last : delimited section (the port) or all the other sections (the whole initial part, the address).

Parameters

- **$name** (*string*) – (from $this->getName() usually)
- **$part** –

Returns string

getIp()
Gets the IP address of the socket

Returns string

getLastError()
Get the last error that occurred on the socket

Returns int|string

isConnected()
Whether the socket is currently connected

Returns boolean

disconnect()
Disconnect the socket

Returns void

getResource()

getResourceId()

send(*$data*)

Parameters

- **$data** (*unknown_type*) –

Returns boolean|int The number of bytes sent or false on error

receive(*$length = self::DEFAULT_RECEIVE_LENGTH*)
Recieve data from the socket

Parameters

- **$length** (*int*) –

Returns string

configureProtocol()
Configures the protocol option

Wrench\Socket\ServerClientSocket

class ServerClientSocket

constant TIMEOUT_SOCKET
Default timeout for socket operations (reads, writes)

constant DEFAULT_RECEIVE_LENGTH

constant NAME_PART_IP
Socket name parts

property socket
protected resource

property context
protected

Stream context

property connected
protected boolean

Whether the socket is connected to a server

Note, the connection may not be ready to use, but the socket is connected at least. See $handshaked, and other properties in subclasses.

property firstRead
protected boolean

Whether the current read is the first one to the socket

property name
protected string

The socket name according to stream_socket_get_name

property options
protected array

property protocol
protected Protocol

__construct(*$accepted_socket*, *$options = array()*)
Constructor

A server client socket is accepted from a listening socket, so there's no need to call ->connect() or whatnot.

Parameters

- **$accepted_socket** (*resource*) –
- **$options** (*array*) –

configure(*$options*)
Configure options

Options include - timeout_connect => int, seconds, default 2 - timeout_socket => int, seconds, default 5

Parameters

- **$options** (*array*) –

Returns void

getName()
Gets the name of the socket

getNamePart(*$name*, *$part*)
Gets part of the name of the socket

PHP seems to return IPV6 address/port combos like this: ::1:1234, where ::1 is the address and 1234 the port So, the part number here is either the last : delimited section (the port) or all the other sections (the whole initial part, the address).

Parameters

- **$name** (*string*) – (from $this->getName() usually)
- **$part** –

Returns string

getIp()
Gets the IP address of the socket

Returns string

getPort()
Gets the port of the socket

Returns int

getLastError()
Get the last error that occurred on the socket

Returns int|string

isConnected()
Whether the socket is currently connected

Returns boolean

disconnect()
Disconnect the socket

Returns void

getResource()

getResourceId()

send(*$data*)

Parameters

- **$data** (*unknown_type*) –

Returns booleanlint The number of bytes sent or false on error

receive(*$length = self::DEFAULT_RECEIVE_LENGTH*)
Recieve data from the socket

Parameters

- **$length** (*int*) –

Returns string

configureProtocol()
Configures the protocol option

Wrench\Socket\ServerSocket

class ServerSocket
Server socket

Used for a server's "master" socket that binds to the configured interface and listens

constant TIMEOUT_SOCKET
Default timeout for socket operations (reads, writes)

constant DEFAULT_RECEIVE_LENGTH

constant NAME_PART_IP
Socket name parts

property listening
protected boolean

Whether the socket is listening

property scheme
protected

property host
protected

property port
protected

property socket
protected resource

property context
protected

Stream context

property connected
protected boolean

Whether the socket is connected to a server

Note, the connection may not be ready to use, but the socket is connected at least. See $handshaked, and other properties in subclasses.

property `firstRead`
protected boolean

Whether the current read is the first one to the socket

property `name`
protected string

The socket name according to stream_socket_get_name

property `options`
protected array

property `protocol`
protected Protocol

`configure`(*$options*)

Parameters

- **`$options`** –

`listen`()
Listens

`accept`()
Accepts a new connection on the socket

Returns resource

`getSocketStreamContextOptions`()

`getSslStreamContextOptions`()

`__construct`(*$uri*, *$options = array()*)
URI Socket constructor

Parameters

- **`$uri`** (*string*) – WebSocket URI, e.g. ws://example.org:8000/chat
- **`$options`** –

`getUri`()
Gets the canonical/normalized URI for this socket

Returns string

`getName`()

`getHost`()
Gets the host name

`getPort`()

`getStreamContext`(*$listen = false*)
Gets a stream context

Parameters

- **`$listen`** –

`getNamePart`(*$name*, *$part*)
Gets part of the name of the socket

PHP seems to return IPV6 address/port combos like this: ::1:1234, where ::1 is the address and 1234 the port So, the part number here is either the last : delimited section (the port) or all the other sections (the whole initial part, the address).

Parameters

- **$name** (*string*) – (from $this->getName() usually)
- **$part** –

Returns string

getIp()
Gets the IP address of the socket

Returns string

getLastError()
Get the last error that occurred on the socket

Returns int|string

isConnected()
Whether the socket is currently connected

Returns boolean

disconnect()
Disconnect the socket

Returns void

getResource()

getResourceId()

send(*$data*)

Parameters

- **$data** (*unknown_type*) –

Returns boolean|int The number of bytes sent or false on error

receive(*$length = self::DEFAULT_RECEIVE_LENGTH*)
Recieve data from the socket

Parameters

- **$length** (*int*) –

Returns string

configureProtocol()
Configures the protocol option

Wrench\Socket\Socket

class Socket
Socket class

Implements low level logic for connecting, serving, reading to, and writing from WebSocket connections using PHP's streams.

Unlike in previous versions of this library, a Socket instance now represents a single underlying socket resource. It's designed to be used by aggregation, rather than inheritence.

constant `TIMEOUT_SOCKET`
Default timeout for socket operations (reads, writes)

constant `DEFAULT_RECEIVE_LENGTH`

constant `NAME_PART_IP`
Socket name parts

property `socket`
protected resource

property `context`
protected

Stream context

property `connected`
protected boolean

Whether the socket is connected to a server

Note, the connection may not be ready to use, but the socket is connected at least. See $handshaked, and other properties in subclasses.

property `firstRead`
protected boolean

Whether the current read is the first one to the socket

property `name`
protected string

The socket name according to stream_socket_get_name

property `options`
protected array

property `protocol`
protected Protocol

`configure`(*$options*)
Configure options

Options include - timeout_connect => int, seconds, default 2 - timeout_socket => int, seconds, default 5

Parameters

- **`$options`** (*array*) –

Returns void

`getName`()
Gets the name of the socket

`getNamePart`(*$name*, *$part*)
Gets part of the name of the socket

PHP seems to return IPV6 address/port combos like this: ::1:1234, where ::1 is the address and 1234 the port So, the part number here is either the last : delimited section (the port) or all the other sections (the whole initial part, the address).

Parameters

- **`$name`** (*string*) – (from $this->getName() usually)
- **`$part`** –

Returns string

getIp()
Gets the IP address of the socket

Returns string

getPort()
Gets the port of the socket

Returns int

getLastError()
Get the last error that occurred on the socket

Returns int|string

isConnected()
Whether the socket is currently connected

Returns boolean

disconnect()
Disconnect the socket

Returns void

getResource()

getResourceId()

send(*$data*)

Parameters

- **$data** (*unknown_type*) –

Returns boolean|int The number of bytes sent or false on error

receive(*$length = self::DEFAULT_RECEIVE_LENGTH*)
Recieve data from the socket

Parameters

- **$length** (*int*) –

Returns string

__construct(*$options = array()*)
Configurable constructor

Parameters

- **$options** –

configureProtocol()
Configures the protocol option

Wrench\Socket\UriSocket

class UriSocket

constant TIMEOUT_SOCKET
Default timeout for socket operations (reads, writes)

constant `DEFAULT_RECEIVE_LENGTH`

constant `NAME_PART_IP`
Socket name parts

property `scheme`
protected

property `host`
protected

property `port`
protected

property `socket`
protected resource

property `context`
protected

Stream context

property `connected`
protected boolean

Whether the socket is connected to a server

Note, the connection may not be ready to use, but the socket is connected at least. See $handshaked, and other properties in subclasses.

property `firstRead`
protected boolean

Whether the current read is the first one to the socket

property `name`
protected string

The socket name according to stream_socket_get_name

property `options`
protected array

property `protocol`
protected Protocol

`__construct` (*$uri*, *$options = array()*)
URI Socket constructor

Parameters

- **`$uri`** (*string*) – WebSocket URI, e.g. ws://example.org:8000/chat
- **`$options`** –

`getUri` ()
Gets the canonical/normalized URI for this socket

Returns string

`getName` ()

`getHost` ()
Gets the host name

`getPort` ()

getStreamContext(*$listen = false*)
Gets a stream context

Parameters

- **$listen** –

getSocketStreamContextOptions()
Returns an array of socket stream context options

See http://php.net/manual/en/context.socket.php

Returns array

getSslStreamContextOptions()
Returns an array of ssl stream context options

See http://php.net/manual/en/context.ssl.php

Returns array

configure(*$options*)
Configure options

Options include - timeout_connect => int, seconds, default 2 - timeout_socket => int, seconds, default 5

Parameters

- **$options** (*array*) –

Returns void

getNamePart(*$name*, *$part*)
Gets part of the name of the socket

PHP seems to return IPV6 address/port combos like this: ::1:1234, where ::1 is the address and 1234 the port So, the part number here is either the last : delimited section (the port) or all the other sections (the whole initial part, the address).

Parameters

- **$name** (*string*) – (from $this->getName() usually)
- **$part** –

Returns string

getIp()
Gets the IP address of the socket

Returns string

getLastError()
Get the last error that occurred on the socket

Returns int|string

isConnected()
Whether the socket is currently connected

Returns boolean

disconnect()
Disconnect the socket

Returns void

getResource()

getResourceId()

send(*$data*)

Parameters

- **$data** (*unknown_type*) –

Returns booleanlint The number of bytes sent or false on error

receive(*$length = self::DEFAULT_RECEIVE_LENGTH*)
Recieve data from the socket

Parameters

- **$length** (*int*) –

Returns string

configureProtocol()
Configures the protocol option

Wrench\Util

Wrench\Util\Configurable

class Configurable
Configurable base class

property options
protected array

property protocol
protected Protocol

__construct(*$options = array()*)
Configurable constructor

Parameters

- **$options** –

configure(*$options*)
Configures the options

Parameters

- **$options** (*array*) –

configureProtocol()
Configures the protocol option

Wrench\Util\Ssl

class Ssl

generatePemFile(*$pem_file*, *$pem_passphrase*, *$country_name*, *$state_or_province_name*, *$locality_name*, *$organization_name*, *$organizational_unit_name*, *$common_name*, *$email_address*)
Generates a new PEM File given the informations

Parameters

- **$pem_file** (*string*) – the path of the PEM file to create
- **$pem_passphrase** (*string*) – the passphrase to protect the PEM file or if you don't want to use a passphrase
- **$country_name** (*string*) – the country code of the new PEM file. e.g.: EN
- **$state_or_province_name** (*string*) – the state or province name of the new PEM file
- **$locality_name** (*string*) – the name of the locality
- **$organization_name** –
- **$organizational_unit_name** –
- **$common_name** –
- **$email_address** (*string*) – the email address

CHAPTER 6

Authors

The original maintainer and author was @nicokaiser. Plentiful improvements were contributed by @lemmingzshadow and @mazhack. Parts of the Socket class were written by Moritz Wutz. The server is licensed under the WTFPL, a free software compatible license.

Index

Symbols

A

B

C

D

E

F

G

H

T

U

V