WebSocket

The `Websocket` class implements an RFC 6455 (https://tools.ietf.org/html/rfc6455) compatible WebSocket server with event handling.

Namespace: `\CLI`
File location: `lib/cli/ws.php`

Constructor

```
$server = new \CLI\WS( $addr, resource $ctx = NULL, $wait = 60 );
```

The first argument `$addr` specifies where the server will be listening for connections. The optional `$ctx` argument accepts the context for the socket and the `$wait` argument defines the timeout value.

Examples

- Listening on the local machine without encryption

```
$server = new CLI\WS('tcp://127.0.0.1:9000');
```

- Listening on all IP addresses with encryption

```
$server = new CLI\WS(
    'ssl://0.0.0.0:9000',
    stream_context_create([
        'ssl'=>[
            'local_cert' => '/path/to/.pem',
            'verify_peer' => FALSE,
            'verify_peer_name' => FALSE,
            'allow_self_signed' => TRUE
        ]
    ])
);
```

Methods

agents

Get all agents with a matching URI.

```
Agent[] agents ( string $uri = NULL )
```

The returned agents can be filtered with the optional `$uri` argument.

events

Return the registered event handlers.

```
array events( )
```

on

Bind a function to an event.

```
\CLI\WS on ( string $event, callable $func )
```

Binds the `$func` hook to the event `$event` and returns itself. Existing bindings for `$event` are replaced by `$func` .

The section Event Handlers (websocket#event-handlers) lists all events.

kill

Properly terminate the server.

```
void kill ( )
```

The shutdown process triggers the `stop` event (websocket#stop) and terminates the PHP process.

run

Execute the server and listen for connections.

```
void run ( )
```

The `run()` method neither returns a value nor terminates. Instead, the application is shut down on request (with the kill() (websocket#kill) method or OS signal) or when an error occurs.

Event Handlers

`Websocket` provides multiple events for customization. Event handlers are registered with the `on()` function (websocket#on), e.g. the following example registers an anonymous function for the `start` event.

```
$server->on('start', function(){
    echo 'Server started';
});
```

start

The `start` hook is executed when the server is started.

```
callable start ( \CLI\WS $server )
```

The event handler receives the server as argument.

error

The `error` hook is executed when an error occurs.

```
callable error ( \CLI\WS $server )
```

The event handler receives the server as argument.

stop

The `stop` hook is executed when the server (incl. the PHP process) is shutting down (https://www.php.net/manual/en/function.register-shutdown-function.php).

```
callable stop ( \CLI\WS $server )
```

The event handler receives the server as argument.

connect

The `connect` hook is executed when a client connects to a server.

```
callable connect ( \CLI\Agent $agent )
```

The event handler receives the new client's agent (websocket#agent) as argument.

disconnect

The `disconnect` hook is executed when a client disconnects from a server.

```
callable disconnect ( \CLI\Agent $agent )
```

The event handler receives the disconnecting client's agent (websocket#agent) as argument. The agent gets destroyed as soon as the handler terminates.

idle

The `idle` hook is executed when a client idles.

```
callable idle ( \CLI\Agent $agent )
```

The event handler receives the client's agent (websocket#agent) as argument.

receive

The `receive` hook is executed when application data is received from a client.

```
callable receive ( \CLI\Agent $agent, $op, $data )
```

The event handler is only called for binary and text frames (see RFC 6455, Section-11.8 (https://tools.ietf.org/html/rfc6455#section-11.8)). Therefore, the event handler receives the operation codes `$op = WS::Binary` (`0x01`) and `$op = WS::Text` (`0x02`).

The event handler receives the client's agent (websocket#agent) as first argument. The operation code `$op` is either `WS::Binary` or `WS::Text`. `$data` contains the application-specific payload from the client.

send

The `send` hook is executed when application data is sent to a client.

```
callable send ( \CLI\Agent $agent, $op, $data )
```

The event handler is not called for the pong (`WS::Pong` , `0x0a`) and connection close (`WS:Close` , `0x08`) frames. Valid operation codes are `WS::Binary` and `WS::Text` (see RFC 6455, Section-11.8 (https://tools.ietf.org/html/rfc6455#section-11.8)).

The event handler receives the client's agent (websocket#agent) as first argument. The operation code `$op` is user-defined (see the section about Agent (websocket#agent)) and should be either `WS::Binary` or `WS::Text`. `$data` contains the application-specific payload for the client.

Agent

server

Get the server of the agent.

```
\CLI\WS server ( )
```

id

Get the socket ID of the agent.

```
string id ( )
```

socket

Get the socket resource of the agent.

```
resource socket ( )
```

verb

Get the HTTP verb of the agent.

```
string verb ( )
```

uri

Get the HTTP URI of the agent.

```
string uri ( )
```

headers

Get the HTTP headers of the agent.

```
string[] headers ( )
```

send

Send application-specific payload to the client of the agent.

```
string send ( $op, $data = '' )
```

Valid operation codes `$op` for application frames are `WS::Binary` and `WS::Text` (see RFC 6455, Section-11.8 (https://tools.ietf.org/html/rfc6455#section-11.8)). `$data` contains the application-specific payload for the client. This method triggers the `send` event (websocket#send). It returns the sent data.

fetch

This is an internal method and should be not called manually.

Examples

A complete WebSocket-based chat application is available at github.com/F3Community/websocket-example (https://github.com/F3Community/websocket-example).