

The provided PHP code sets up a server using ReactPHP, a library that allows for asynchronous network programming in PHP. Here's a step-by-step breakdown of what it does:

1. Autoload Dependencies:

php

```
require 'vendor/autoload.php';
```

2. Use Statements for Required Classes:

php

```
use React\Http\HttpServer;
use React\Http\Message\Response;
use
Psr\Http\Message\ServerRequestInterface;
use React\EventLoop\Factory;
use React\Socket\Server as SocketServer;
// use
Yacht\RetainedStorage\implementation\File;
```

3. Function to Pretty-Print Variables: This function provides a human-readable format for variable output by using JSON encoding with specific flags.

php

```
function pretty ($var) {
    return gettype($var) . ' ' .
    json_encode($var, JSON_UNESCAPED_SLASHES |
```

```
JSON_UNESCAPED_UNICODE | JSON_PRETTY_PRINT |
JSON_PARTIAL_OUTPUT_ON_ERROR |
JSON_INVALID_UTF8_SUBSTITUTE);
}
```

4. Initialization of Data Arrays: The code initializes arrays to store different types of data including weather, wind, waves, and GPS data.

php

```
$data = [ 'weather' => [], 'wind' => [],
'waves' => [], 'gps' => [] ];
```

5. Function to Fetch GPS Data: This function connects to a GPSD server (presumably running on localhost at port 2947) and reads GPS data. It updates the `$data['gps']` array with this information.

php

```
function fetchGpsData(&$data) {
    $fp = fsockopen('127.0.0.1', 2947,
$errno, $errstr, 30);
    if (!$fp) {
        echo "ERROR: $errno -
$errstr<br />\n";
    } else {
        fwrite($fp, '?
WATCH={"enable":true,"json":true}');
        while (!feof($fp)) {
            $line = fgets($fp, 128);
```

```

        $json = json_decode($line,
true);
        if (isset($json['class']) &&
$json['class'] === 'TPV') {
            $data['gps'] = [
                'lat' => $json['lat']
?? 'n/a',
                'lon' => $json['lon']
?? 'n/a',
                'alt' => $json['alt']
?? 'n/a',
                'speed' =>
$json['speed'] ?? 'n/a',
                'track' =>
$json['track'] ?? 'n/a',
                'time' =>
$json['time'] ?? 'n/a',
            ];
            break;
        }
    }
    fclose($fp);
}
}

```

6. Event Loop and Storage Initialization: It sets up an event loop using ReactPHP and initializes a file-based retained storage system to persist data.

php

```
$loop = Factory::create();
```

```

    $storage = new \Yacht\RetainedStorage\implementation\File(
        __DIR__ . '/storage', 10); // 10 MB max size
    $dataKey = 'data';
    $data = $storage->retrieve($dataKey) ?: [
        'weather' => [], 'wind' => [], 'waves' =>
        [], 'gps' => [] ];

```

7. HTTP Server Setup: Defines an HTTP server that listens for specific routes to either update GPS data or retrieve current data:

php

```

    $server = new HttpServer(function
(ServerRequestInterface $request) use
(&$data, $storage, $dataKey) {
        $path = $request->getUri()-
>getPath();
        if ($path === '/update') {
            fetchGpsData($data);
            $storage->save($dataKey, $data);
            return new Response(200,
['Content-Type' => 'text/plain'], 'Data
updated!');
        } elseif ($path === '/data') {
            return new Response(200,
['Content-Type' => 'application/json'],
json_encode($data));
        }
        return new Response(404, ['Content-
Type' => 'text/plain'], 'Not Found');
    });

```

8. Run the Server: Binds the server to all network interfaces on port 5000 and starts the event loop.

php

```
$socket = new  
SocketServer('0.0.0.0:5000', $loop);  
$server->listen($socket);  
echo "Server running at http://  
0.0.0.0:5000\n";  
$loop->run();
```

In essence, this PHP script sets up a basic ReactPHP server that can update and retrieve GPS data through specific endpoints [\[1\]](#), [\[2\]](#), [\[3\]](#), [\[4\]](#), [\[5\]](#).

Sources

srv.pdf

1, 2, ..., 5.

srv.pdf

BQ

Does function fetchGpsData open the connection, read a line of data from the gps and then disconnect and returns? Or does it read data continuously, not returning, just update the data in the global variable \$data?



Searching file srv.pdf for: fetchGpsData

Dit dus, knap ;)

The function `fetchGpsData` opens a connection, reads lines of data from the GPSD server and then disconnects. It does not read data continuously; it reads data within a loop until it finds the information it needs and then updates the global variable `$data['gps']` with the GPS data, after which it breaks out of the loop and closes the connection.

Here's how it works step-by-step:

1. Open a Connection: The function uses `fsockopen` to connect to the GPSD server running on `127.0.0.1` at port `2947`. If the connection fails, it prints an error message:

php

```
$fp = fsockopen('127.0.0.1', 2947,
$errno, $errstr, 30);
if (!$fp) {
    echo "ERROR: $errno - $errstr<br /
>\n";
}
```

2. Send a Command: If the connection is successful, it sends a command to enable JSON reporting via the `fwrite` function:

php

```
else {
    fwrite($fp, '?
WATCH={"enable":true,"json":true}');
}
```

3. Read Data in a Loop: The function then enters a `while` loop where it reads lines from the connection using `fgets`:

php

```
while (!feof($fp)) {
    $line = fgets($fp, 128);
}
```

```

        $json = json_decode($line, true);
        if (isset($json['class']) &&
$json['class'] === 'TPV') {
            $data['gps'] = [
                'lat' => $json['lat'] ?? 'n/a',
                'lon' => $json['lon'] ?? 'n/a',
                'alt' => $json['alt'] ?? 'n/a',
                'speed' => $json['speed'] ??
'n/a',
                'track' => $json['track'] ??
'n/a',
                'time' => $json['time'] ??
'n/a',
            ];
            break;
        }
    }
}

```

4. Update Data and Close Connection: Once it finds the required TPV (Time-Position-Velocity) data class, it updates the `$data['gps']` array and breaks out of the loop to close the connection using `fclose`:

php

```

        fclose($fp);
    }
}

```


Thus, the function `fetchGpsData` reads GPS data until it finds the TPV class, updates the `&data['gps']` global variable, and then closes the connection. It does not read data continuously ad infinitum.

Please let me know if you'd like more information or details about specific parts of the code [\[1\]](#), [\[2\]](#), [\[3\]](#).