# PScript: The Documentation

University of Turku
Department of Computing
Master of Science Thesis
Computer science
July 2024
Lassi Haapala

# Contents

# 1 Introduction

PScript is a multi language programming setup, meant to extend the usability and metaprogrammability of the PHP-JavaScript relationship. PHP and JavaScript share a meta-level relationship due to the their server to client relationship. PHP is able to access and modify the source code of JavaScript before it delivers finalized payload to the client-side browser for execution. This ability enables PHP to metaprogram JavaScript, however, the relationship is often too unreliable to see wide spread usage.

The mentality behind PScript is to improve upon the doubts arising from the unreliable nature of the existing relationship. PScript achieves these improvements by adding an additional pseudo runtime inside the existing PHP host in a preprocessor format. The additional runtime provides capabilities for additional syntax definitions and other improvements of the relationship. Therefore, the main goals of each improvement and feature are the usability, efficiency, security and metaprogrammability of the existing language relationship.

This paper is meant to act as the complimentary documentation to the main thesis describing the PScript implementation. The following will therefore describe the full-development setup for the PScript environment. Additionally, the documentation will describe the perquisites for running the docker based environment and the development of additional PScript features.

# 2 Project setup

The PScript development environment operates mainly using the docker and docker-compose services. This chapter will run you through the steps required to operate the docker based development environment.

## 2.1 Cloning the project

Before the project can be used one must naturally clone the source to the local host environment. This occurs easily using the following git command `git clone git@github.com:lphaap/php-script.git`.

## 2.2 Docker installation

The first and only perquisite of the PScript environment remains the Docker container engine. Depending on your system docker can be installed in various ways. Please follow the correct guide from `https://docs.docker.com/engine/install/`. Once installed verify installation using `docker version` command. The command should print the latest Docker version.

Docker itself consists of many sub-utilities. From these utilities PScript employs mainly `docker buildx` for building the runnable sandbox images and `docker compose` for coordinating the deployment of multiple sandboxes. Depending on the installation method of the main Docker package, the aforementioned utilities can be

already be included. Check for existing installations using the commands `docker buildx version` and `docker compose version`, which should respond much like the previous docker command. If either of the packages are missing, see the following guides; `https://docs.docker.com/compose/install/` and `https://docs.docker.com/build/architecture/#buildx`

A final good to have utility from the docker family is the Docker desktop GUI. It provides insight to the currently running containers and makes debugging new features easier. Read more here; `https://www.docker.com/products/docker-desktop/`

## 2.3 Bash scripts and commands

After the installation of docker is verified and the project available in the local filesystem, the PScript project environment can be operated fully with custom made bash scripts. The two scripts are located in the root of the project and should also be ran directly from there.

The `./start.sh` command initiates the `docker buildx` module by first pulling and the building the required container images. These images contain the `Nginx` web-server and the `PHP-FPM` interpreter. The script then proceeds to start the containers according to the configuration file in `/docker/docker-compose.yml`. After the command finishes the project setup is complete.

The `./stop.sh` command is naturally the opposite to the start command. It removes all PScript containers that were previously instantiated. This can be useful while debugging as a full rebuilt of the project can often solve issues in docker environments.

Useful commands for docker debugging and execution; Use `docker ps -a` to see all running containers after the start script is ran. Use `docker image prune` to remove cached image builds. Use `docker logs -f` to follow the error and access logs from the PScript environment

After the setup is complete the PScript navigator is available at `localhost:8000`.

# 3 Environment

After the setup is complete one might find the need to adjust the environment for further needs. The `/docker/docker-compose.yml` contains all relevant configurations for the dockerized environment. This file can be used to extend the environment variables and other execution specifics of the PScript development environment. In accordance to the configuration file the `/docker/nginx.conf` file can also be adjusted to modify the configuration of the Nginx web-server container.

Otherwise the PScript environment operates according to a simple index routing logic of the PHP host. The `/src/index.php` displays this logic together with one environment variable `DEBUG`. The `DEBUG` variable can be set to true to display errors from the PHP interpreter which are otherwise automatically handled by the host.

# 4 PScript

The PScript runtime is contained fully inside the `/src/processor/` directory. It contains three utility classes; `Context.php, PSCriptBlock.php, PScriptVar.php`. The `Context` class is a helper class containing the current variables evaluated by the PScript environment. The class is mainly used for the functionality of active variable transfers.

The `PScriptBlock` and `PScriptVar` are utility classes used for storing parsed PScript references and `client` blocks. The `PScriptVar` can be used to encapsulate a single `client` reference, while the `PScriptBlock` can be used to store a dynamically parsed PScript template or code block.

Finally the `PScript.php` file contains the preprocessor runtime itself. If one desires to extend the functionalities of PScript this file is the place to do so. The main `parse` function can be modified in order to accommodate additional syntax definitions into the main processing algorithm. On the other hand the `convert_variable` function can be used to support additional variable transformations between PHP and JavaScript.