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 stage programming language, designed to improve the 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 PHP-JavaScript 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 existing relationship. The main goals of these improvements are the usability, efficiency, security and metaprogrammability of the PHP-JavaScript relationship.

This documentation is meant to act as the complimentary piece 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 configured one must clone the source files to the local host environment. This can be achieved by using the git version-control tool, i.e., `git clone git@github.com:lphaap/php-script.git`.

2.2 Docker installation

The first and only perquisite of the PScript environment is 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 the 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 the `docker buildx` utility for building runnable container images and the `docker compose` utility for coordinating the deployment of multiple docker containers. 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 version` command. If either of the packages are missing, see the following guides for installation details; `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 is cloned to the local filesystem, the PScript project environment can be operated with custom made bash scripts. The two scripts are located in the root of the cloned 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 prebuilt images according to the configuration file in `/docker/docker-compose.yml`. After the `./start.sh` command finishes the project setup is completed.

The `./stop.sh` command is the opposite of the start command. It removes all PScript containers that were previously instantiated. This command can be useful both for cleaning up the existing environment and for debugging it. A full rebuild of the project environment can often solve issues resulting from the docker utilities themselves.

Some useful commands for docker based development include but are not limited

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

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

3 Environment

After the PScript setup is completed, 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 with additional variables and other execution specifics provided to the PScript development environment.

The PScript development environment operates based on a Nginx web-server. The web-server is configured in the configuration file `/docker/nginx.conf`. The file can be adjusted to modify the configuration of the Nginx web-server container to suit the needs of further development.

The development PHP host operates according to a simple index routing logic located in the PHP file `/src/index.php`. This file can be modified to extend the navigation and routing logic of the PScript environment. Additionally the error reporting logic of the project is based on 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. The directory contains three utility classes; `Context.php`, `PSCriptBlock.php`, `PScript-Var.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 can be modified to achieve this. For instance, the `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.