Festo Workstation – Program File Overview

Overview
This document briefly describes the purpose of every relevant software file used in the Festo workstation control system. The system consists of backend C++ programs running on the ZedBoard, web-based HTML interfaces, and PHP scripts connecting the browser to the hardware.

Backend C++ Programs (running on ZedBoard)

main.cpp
Entry point of the backend program. Runs an infinite loop every 0.1 seconds to read sensors, validate data, apply safety logic, detect faults, and generate JSON system status data.

Data.cpp / Data.hpp
Defines a data container class that stores all sensor readings and actuator states. Provides getter and setter functions but contains no hardware access logic.

DataFilter.cpp / DataFilter.hpp
Reads raw sensor and actuator values from FPGA shared memory. Checks whether sensor values lie within valid physical ranges. Stores validated values in the Data object.

ErrorDetector.cpp / ErrorDetector.hpp
Monitors recent trends of tank level and valve states. Detects persistent abnormal behavior and generates diagnostic error codes (leakage or blockage). Does not control actuators, only reports faults.

ConvertAndPrepareData.cpp / ConvertAndPrepareData.hpp
Converts raw sensor readings into engineering units. Applies safety shutdown rules (over-temperature, empty tank, overflow). Generates JSON formatted system status and writes it to a text file. Also writes binary data for external applications.

Compiled Executable
The compiled result of the C++ files is a binary program named *getData*. It runs continuously in the background on the ZedBoard.

Generated Data File
json_data.txt
Text file continuously updated by the backend program. Contains the latest system status in JSON format.

Web-Based Monitoring and Control Files

get_data.html
Webpage for live monitoring. Requests JSON data every 0.5 seconds and displays sensor values, tank levels, actuator states, warnings, and errors in the browser.

post_data.html
Webpage for manual control. Provides buttons to switch pump, valves, and heater ON or OFF by

sending HTTP requests to PHP scripts.

stylesheet.css
Defines the visual layout and style of the monitoring and control webpages.

PHP Interface Scripts

get_data.php
Reads the latest json_data.txt file and sends its content to the browser as the HTTP response for live monitoring.

post_pump.php
Receives pump control commands from the browser and writes them into FPGA shared memory to control the pump state.

post_valveUpper.php
Receives commands to open or close the upper valve and writes them into FPGA shared memory.

post_valveLower.php
Receives commands to open or close the lower valve and writes them into FPGA shared memory.

post_heater.php
Receives heater ON/OFF commands from the browser and writes them into FPGA shared memory.

post_filter.php
Receives filter ON/OFF commands (optional subsystem).

post_inflow.php
Receives inflow ON/OFF commands (optional subsystem).

reboot.php
Triggers a system reboot command on the ZedBoard through the web interface.

shutdown.php
Triggers a system shutdown command on the ZedBoard through the web interface.

shutdown_file
Auxiliary file used as a marker for shutdown operations in the system.

Overall Summary
The backend C++ program continuously reads hardware sensors, applies safety and fault logic, and publishes system status. The HTML pages provide user interfaces for monitoring and control. The PHP scripts bridge browser commands to hardware-level memory operations on the ZedBoard.