FW Framework

Comprehensive C++ Application Framework

Project Overview

The FW Framework is a comprehensive C++ application framework designed for system-level programming, web application development, and embedded systems. Developed by Douglas Wade Goodall, this framework provides extensive functionality for environment management, network configuration, system logging, and inter-process communication.

> **Key Purpose:** This framework serves as a foundation for building robust, scalable applications with emphasis on system integration, network protocols, and shared memory management.

Project Statistics

Metric	Value
Total Source Files	50+ C++ files
Core Components	12 major modules
Development Period	2021-2025
Build System	CMake

System Architecture

Framework Components

Environment Manager **Shared Memory**

Network Services **Logging System**

CGI Interface **GPIO Control**

Core Architecture Principles

Modular Design

Each component is self-contained and can be used independently or as part of the larger framework.

Shared Memory

Efficient inter-process communication through shared memory segments with mutex protection.

Environment Awareness

Automatic detection and configuration of system environment, network interfaces, and protocols.

Cross-Platform Support

Designed to work across different Linux distributions and embedded systems.

Component Dependencies

Component	Dependencies	Purpose
mwfw2	Core framework	Main initialization and feature management
environment	shared, osIface	System environment detection and configuration
shared	shMemMgr, shMemMutex	Shared memory management

vparpc	network utilities	Remote procedure call implementation

Key Features

🌐 Environment Management

Automatic detection of hostname, IP addresses, network interfaces, and protocol configuration (HTTP/HTTPS).

📊 System Logging

Comprehensive logging system with CLog and CSysLog classes for application and system-level logging.

🔗 Shared Memory IPC

Efficient inter-process communication using shared memory with mutex synchronization.

🌍 CGI Web Interface

Built-in CGI support for web-based applications with automatic environment detection.

⚡ GPIO Control

Hardware interface for GPIO pin control through RTkGPIO module for embedded applications.

🔧 Schema Management

Dynamic schema compilation and management for database-like operations.

🔒 Security Features

Password management, user authentication, and secure configuration handling.

📝 Configuration System

Flexible configuration management with dotconfig and environment-specific settings.

Network Capabilities

- **Protocol Detection:** Automatic HTTP/HTTPS protocol determination
- **Public IP Discovery:** External IP address detection using curl
- **Network Interface Analysis:** Primary network interface identification
- **Port Management:** xinetd service control and port triggering
- **Remote Procedure Calls:** VPA RPC implementation for distributed services

Environment Management

The environment management system is one of the framework's core components, responsible for automatically detecting and configuring system parameters.

Environment Detection Process

Step	Function	Description
1	extract_username()	Extracts username from source file path
2	get_interface()	Determines primary network interface
3	set_hostname()	Reads system hostname from /etc/hostname
4	set_protocol()	Tests for HTTPS/HTTP availability
5	get_ip()	Discovers local IP address
6	set_public_ip()	Fetches public IP using external service
7	set_*_root()	Configures various directory paths

Directory Structure Configuration

> The framework automatically configures paths for CGI scripts, images, styles, journals, and temporary files based on the detected username and system configuration.

- **CGI Root:** `http://[IP]/~[user]/fw/cgi-bin/`
- **Images Root:** `http://[IP]/~[user]/fw/images/`
- **Styles Root:** `http://[IP]/~[user]/fw/styles/`
- **Journal Root:** `/home/[user]/Documents/Fw Notes/`
- **Config Root:** `/home/[user]/.config/multiware`

- **Temp Root:** `/home/[user]/public_html/fw/tmp/`

API Reference

Environment Class Methods

Method	Return Type	Description
get_public_ip()	char*	Returns the public IP address of the host
get_hostname(bool)	std::string	Retrieves the system hostname
get_interface(bool)	std::string	Gets the primary network interface name
get_ip(bool)	std::string	Fetches the local IP address
is_curl_present()	bool	Checks if cURL is installed on the system
is_netstat_present()	bool	Verifies netstat utility availability

Configuration Methods

Method	Purpose
set_cgi_root(bool)	Configures CGI script directory path
set_img_root(bool)	Sets image resources directory
set_styles_root(bool)	Configures stylesheet directory
set_journal_root(bool)	Sets journal files directory
set_config_root()	Configures user-specific settings directory

Usage Examples

```
  // Initialize environment
environment env;

// Get system information
std::string hostname = env.get_hostname(false);
std::string ip = env.get_ip(false);
char* public_ip = env.get_public_ip();

// Check system capabilities
if (env.is_curl_present()) {
    // Use cURL for network operations
}

// Configure paths
env.set_cgi_root(false);
env.set_img_root(false);
```

Installation & Build

Prerequisites

Required Dependencies:

- C++11 compatible compiler (GCC 7+ or Clang 5+)
- CMake 3.10 or higher
- POSIX-compliant operating system (Linux preferred)
- curl utility for network operations
- netstat utility for network interface detection

Build Instructions

```
# Clone the repository
git clone [repository-url]
cd fw

# Create build directory
mkdir build && cd build

# Configure with CMake
cmake ..

# Build the project
make

# Optional: Install system-wide
sudo make install
```

CMake Configuration Options

Option	Default	Description
CMAKE_BUILD_TYPE	Release	Build configuration (Debug/Release)
ENABLE_GPIO	ON	Enable GPIO functionality
ENABLE_CGI	ON	Enable CGI web interface

Usage Guide

Basic Framework Initialization

```
  #include "mwfw2.h"

int main() {
    // Initialize the framework
    mwfw2* framework = new mwfw2(__FILE__, __FUNCTION__);

    // Framework automatically configures:
    // - Environment variables
    // - Shared memory
    // - Logging system
    // - Network settings

    // Your application code here
```

```
    return 0;
}
```

Environment Usage

```
// Environment is automatically initialized by mwfw2
// Access through global pointer gpEnv

// Get system information
std::string hostname = gpEnv->get_hostname(false);
std::string local_ip = gpEnv->get_ip(false);
char* public_ip = gpEnv->get_public_ip();

// Check capabilities
if (gpEnv->is_curl_present()) {
    // Perform network operations
}
```

Logging Usage

```
// System logging (available globally)
gpSysLog->loginfo("Application started");

// Application logging
gpLog->log("Debug information");

// Framework logging through mwfw2
framework->sl_loginfo("System message");
```

> **Best Practice:** Always initialize the framework first before using any of its components. The framework handles dependency resolution and proper

> initialization order automatically.

Troubleshooting

Common Issues

Issue	Cause	Solution
Build fails with "mwfw2.h not found"	Missing include directory	Ensure include/ directory is in include path
Environment initialization fails	Missing system utilities	Install curl and netstat packages
Shared memory access denied	Insufficient permissions	Run with appropriate user permissions
Network interface not detected	netstat output parsing failure	Check netstat installation and output format

Debug Mode

> Many functions accept a debug parameter. Set to `true` to enable verbose logging for troubleshooting.

```
  // Enable debug output
env.set_ip(true);          // Enables debug for IP detection
env.set_protocol(true);  // Enables debug for protocol detection
```

Log File Locations

- **System logs:** Accessible through syslog (typically /var/log/syslog)
- **Application logs:** Framework-specific location based on configuration
- **Temporary files:** `/home/[user]/public_html/fw/tmp/`

Version Information

Attribute	Value
Project Name	FW Framework
Current Version	4.0.0
Development Period	2021-2025
Author	Douglas Wade Goodall
License	All Rights Reserved
Build System	CMake
Language	C++11

Recent Changes

> **Version 4.0.0 Highlights:**
> - Enhanced environment detection system
> - Improved shared memory management
> - GPIO control integration
> - Expanded CGI functionality
> - Better error handling and logging

Compatibility

- **Operating Systems:** Linux (Ubuntu, CentOS, Debian)
- **Architectures:** x86_64, ARM
- **Compilers:** GCC 7+, Clang 5+
- **Standards:** C++11, POSIX

FW Framework - Comprehensive C++ Application Framework