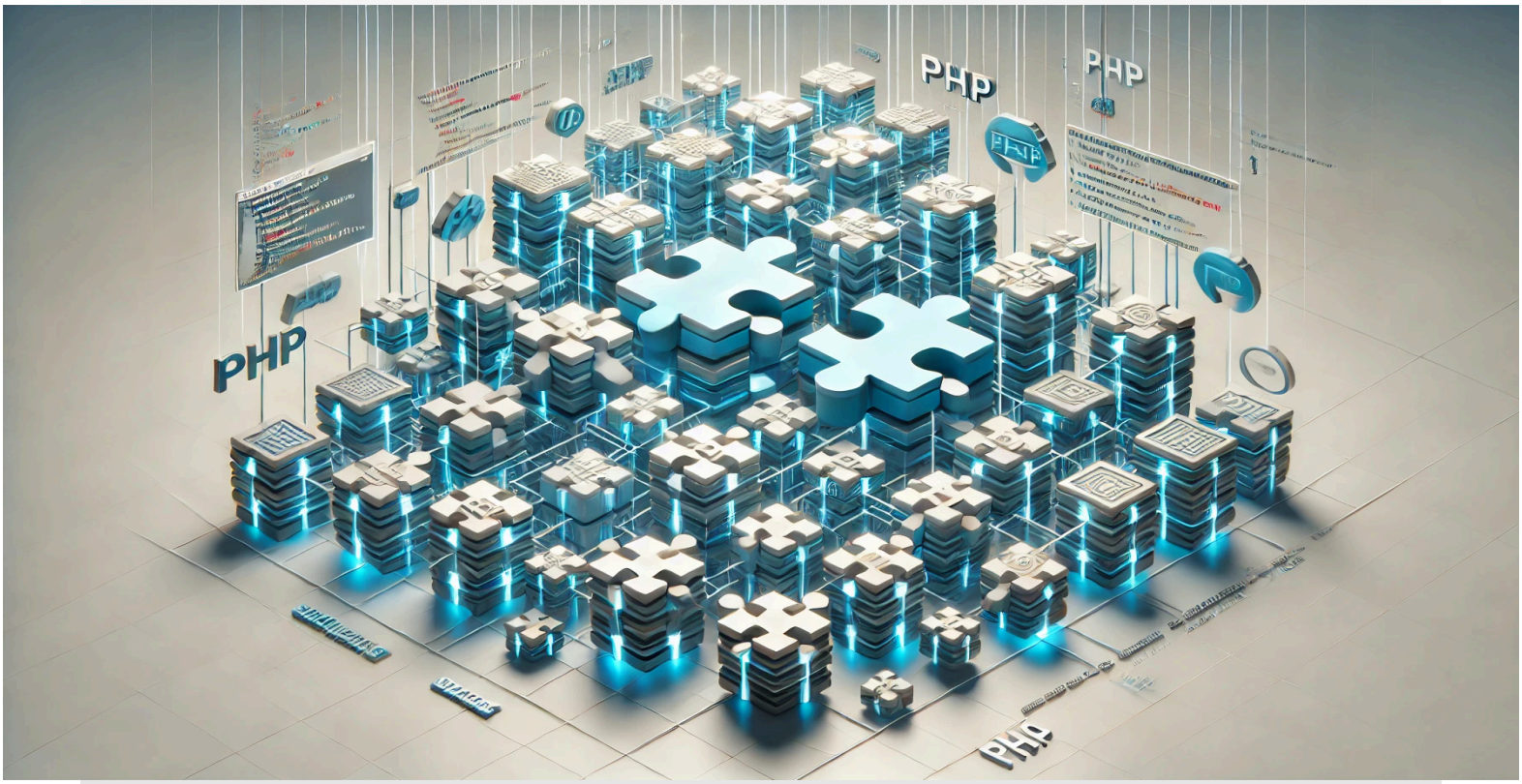
TABLE OF CONTENTS



LET'S TALK - BUILDING A MODULAR PHP FRAMEWORK PART 2

Author(s): Louis Ouellet

Time for part 2! In our previous part, we set up the groundwork of our modular PHP framework. This time, we will focus on expanding its capabilities to support the following objectives:

- Add support for extensions
- Begin implementing a Command-Line module
- Add support for models to be used to create shared methods that require a database
- Add support for helpers to be used to create shared methods that do not require a database

These enhancements will provide the flexibility we need to build modular, maintainable, and extensible applications. Let's walk through each update step by step.

UPDATING THE BOOTSTRAP

We will begin by updating our `Bootstrap` class to include helpers and models. We have also changed the default class for `CLI` to `LaswitchTech\Core\CLI` so that we can start working on it before exporting it into its own module.

Source: Bootstrap.php

src/Bootstrap.php

```php
<?php

/**
 * Core Framework - Bootstrap
 *
 * @license    MIT (https://mit-license.org/)
 * @author     Louis Ouellet <louis@laswitchtech.com>
 */

// Declaring namespace
namespace LaswitchTech\Core;

// Import additionnal class into the global namespace
use LaswitchTech\Core\Config;
use LaswitchTech\Core\Strap;
use Exception;

class Bootstrap {

    const Default = [
        "LOG" => [
            "class" => "\LaswitchTech\Core\Log",
            "scope" => [
                "Router",
                "API",
                "CLI"
            ]
        ],
```

```
        "REQUEST" => [
            "class" => "\LaswitchTech\Core\Request",
            "scope" => [
                "Router",
                "API",
                "CLI"
            ]
        ],
        "OUTPUT" => [
            "class" => "\LaswitchTech\Core\Output",
            "scope" => [
                "Router",
                "API",
                "CLI"
            ]
        ],
        "NET" => [
            "class" => "\LaswitchTech\coreNet\Net",
            "scope" => [
                "Router",
                "API",
                "CLI"
            ]
        ],
        "HELPER" => [
            "class" => "\LaswitchTech\Core\Helpers",
            "scope" => [
                "Router",
                "API",
                "CLI"
            ]
        ],
        "DATABASE" => [
            "class" => "\LaswitchTech\coreDatabase\Database",
            "scope" => [
                "Router",
                "API",
                "CLI"
            ]
        ],
        "MODEL" => [
            "class" => "\LaswitchTech\Core\Models",
            "scope" => [
```

```
                "Router",
                "API",
                "CLI"
            ]
        ],
        "LOCALE" => [
            "class" => "\LaswitchTech\coreLocale\Locale",
            "scope" => [
                "Router",
                "API",
                "CLI"
            ]
        ],
        "ENCRYPTION" => [
            "class" => "\LaswitchTech\coreEncryption\Encryption",
            "scope" => []
        ],
        "CSRF" => [
            "class" => "\LaswitchTech\coreCSRF\CSRF",
            "scope" => [
                "Router",
                "API"
            ]
        ],
        "SLS" => [
            "class" => "\LaswitchTech\coreSLS\SLS",
            "scope" => [
                "Router",
                "API",
                "CLI"
            ]
        ],
        "SMS" => [
            "class" => "\LaswitchTech\coreSMS\SMS",
            "scope" => [
                "Router",
                "API",
                "CLI"
            ]
        ],
        "SMTP" => [
            "class" => "\LaswitchTech\coreSMTP\SMTP",
```

```
            "scope" => [
                "Router",
                "API",
                "CLI"
            ]
        ],
        "IMAP" => [
            "class" => "\LaswitchTech\coreIMAP\IMAP",
            "scope" => [
                "Router",
                "API",
                "CLI"
            ]
        ],
        "INSTALLER" => [
            "class" => "\LaswitchTech\coreInstaller\Installer",
            "scope" => [
                "Router",
                "API",
                "CLI"
            ]
        ],
        "AUTH" => [
            "class" => "\LaswitchTech\coreAuth\Auth",
            "scope" => [
                "Router",
                "API"
            ]
        ],
        "ROUTER" => [
            "class" => "\LaswitchTech\coreRouter\Router",
            "scope" => [
                "Router"
            ]
        ],
        "API" => [
            "class" => "\LaswitchTech\coreAPI\API",
            "scope" => [
                "API"
            ]
        ],
        "CLI" => [
            "class" => "\LaswitchTech\Core\CLI",
```

```
            "scope" => [
                "CLI"
            ]
        ]
    ];

    /**
     * Constructor.
     */
    public function __construct($scope){

        // Set the global variable
        global $CONFIG;

        // Initialize Config
        $CONFIG = new Config(['bootstrap']);

        // Retrieve the straps
        $straps = $CONFIG->get('bootstrap');

        // Loop through the straps
        foreach(self::Default as $strap => $config){

            // Check if an alternate strap exist
            if(isset($straps[$strap])){

                // Loop through the alternate strap
                foreach($config as $key => $value){

                    // Check if the alternate strap has the key
                    if(isset($straps[$strap][$key])){

                        // Set the alternate strap
                        $config[$key] = $straps[$strap][$key];
                    }
                }
            }

            // Check if strap is not the scope
            if(!in_array($scope,$config['scope'])) continue;

            // Initialize the Global Variable
```

```
            global ${$strap};

            // Set Class
            $class = $config['class'];

            // Check if the class exists
            if(class_exists($class)){

                // Initialize the class in the global namespace
                ${$strap} = new $class();
            } else {

                // Set default to null
                ${$strap} = new Strap();
            }
        }
    }
}
```

UPDATING THE REQUEST CLASS

Next, let's update the `Request` class so it can handle command-line arguments. In the constructor, we store the global `$argv` in an `Arguments` property (with a default empty array).

Source: Request.php

src/Request.php

```
<?php

/**
 * Core Framework - Request
 *
 * @license    MIT (https://mit-license.org/)
 * @author     Louis Ouellet <louis@laswitchtech.com>
 */
```

```php
// Declaring namespace
namespace LaswitchTech\Core;

// Import additionnal class into the global namespace
use Exception;

class Request {

    // Properties
    private $Get;
    private $Post;
    private $Files;
    private $Server;
    private $Cookie;
    private $Request;
    private $Arguments;

    /**
     * Constructor
     */
    public function __construct(){

        // Import Global Variables
        global $_GET, $_POST, $_FILES, $_SERVER, $_COOKIE, $_REQUEST,
$argv;

        // Set the properties
        $this->Get = $_GET;
        $this->Post = $_POST;
        $this->Files = $_FILES;
        $this->Server = $_SERVER;
        $this->Cookie = $_COOKIE;
        $this->Request = $_REQUEST;
        $this->Arguments = $argv ?? [];

        // Unset the global variables
        unset($_SERVER, $_GET, $_POST, $_FILES, $_COOKIE, $_REQUEST,
$argv);
    }

    /**
     * Get the host
```

```
     */
    public function getHost() {
        return $this->Server['HTTP_HOST'] ?? '';
    }

    /**
     * Get the host
     */
    public function getHostSSL() {
        return $this->Server['HTTPS'] == 'on' ? 'https://' : 'http://';
    }

    /**
     * Get the host address
     */
    public function getHostAddress() {
        return defined('STDIN') ? 'localhost' : $this->getHostSSL() .
$this->getHost();
    }

    /**
     * Get the URI
     * @return string
     */
    public function getUri() {
        return parse_url($this->Server['REQUEST_URI'] ?? '',
PHP_URL_PATH);
    }

    /**
     * Get the URI segments
     * @return array
     */
    public function getUriSegments() {
        return array_filter(explode( '/', $this->getUri()));
    }

    /**
     * Set and Return the Namespace property
     */
    public function getNamespace(){
        $namespace = "";
        foreach($this->getUriSegments() as $Segment){
```

```
            $namespace .= "/{$Segment}";
        };
        return $namespace;
    }

    /**
     * Get the request method
     * @return string
     */
    public function getMethod() {
        return $this->Server["REQUEST_METHOD"] ?? 'GET';
    }

    /**
     * Get the query string parameters
     * @return string
     */
    public function getQueryString() {
        return $this->Server['QUERY_STRING'] ?? '';
    }

    /**
     * Get the parameters
     * @param string $type
     * @param string $key
     * @return mixed
     */
    public function getParams($type, $key = null){
        switch(strtoupper($type)){
            case 'GET':
                $array = $this->Get;
                break;
            case 'POST':
                $array = $this->Post;
                break;
            case 'FILES':
                $array = $this->Files;
                break;
            case 'COOKIE':
                $array = $this->Cookie;
                break;
            case 'SERVER':
```

```
                $array = $this->Server;
                break;
            case 'QUERY':
                parse_str($this->Server['QUERY_STRING'], $array);
                break;
            case 'REQUEST':
                $array = $this->Request;
                break;
            default:
                return null;
        }
        return !is_null($key) ? ($array[$key] ?? null) : $array;
    }

    /**
     * Decode the REQUEST data
     * @param string $string
     * @return string
     */
    public function decode($string){
        return urldecode(base64_decode($string));
    }

    /**
     * Get the arguments
     * @param int $index
     * @return mixed
     */
    public function getArguments($index = null){
        return !is_null($index) ? ($this->Arguments[$index] ?? null) :
$this->Arguments;
    }

    /**
     * Request for input
     * @param string $string
     * @param array|int|string $options
     * @param string $default
     * @return string
     */
    public function request($string, $options = null, $default = null){
        if(defined('STDIN')){
            $modes = ['select','text','string'];
```

```
            $mode = 'string';
            if($options != null || $options == 0){
                if(is_array($options)){
                    $mode = 'select';
                } else if(is_int($options)){
                    $mode = 'text';
                } else {
                    if(is_string($options)){ $default = $options; }
                }
            }
            $stdin = function(){
                $handle = fopen ("php://stdin","r");
                return str_replace("\n",'',fgets($handle));
            };
            switch($mode){
                case"select":
                    $answer = null;
                    foreach($options as $key => $value){
                        $options[$key] = strtoupper($value);
                    }
                    while($answer == null ||
!in_array(strtoupper($answer),$options)){
                        print_r($string . ' (');
                        foreach($options as $key => $option){
                            if($key > 0){ print_r('/'); }
                            print_r($option);
                        }
                        print_r(')');
                        if($default != null){ print_r('['.$default.']');
}
                        print_r(': ');
                        $answer = $stdin();
                        if($default != null && $answer == ""){ $answer =
$default; }
                    }
                    break;
                case"text":
                    $answer = '';
                    $exits = ['END','EXIT','QUIT','EOF',':Q',''];
                    $count = 0;
                    $max = 5;
                    $print = false;
```

```
                    if(is_bool($default)){ $print = $default; }
                    if(is_int($options)){ $max = $options; }
                    if($print){
                        print_r($string . ' type (');
                        foreach($exits as $key => $exit){
                            if($key > 0){ print_r('/'); }
                            print_r($exit);
                        }
                        print_r(') to exit' . PHP_EOL);
                    } else {
                        print_r($string . PHP_EOL);
                    }
                    do {
                        $line = fgets(STDIN);
if(in_array(strtoupper(str_replace("\n",'',$line)),$exits)){
                            if($max <= 0){ $max = 1; }
                            $count = $max;
                        } else { $answer .= $line; $count++; }
                    } while ($count < $max || $max <= 0);
                    break;
                default:
                    $answer = null;
                    while($answer == null){
                        print_r($string . ' ');
                        if($default != null){ print_r('['.$default.']');
}
                        print_r(': ');
                        $answer = $stdin();
                        if($default != null && $answer == ""){ $answer =
$default; }
                        if($answer == ''){ $answer = null; }
                    }
                    break;
            }
            $answer = trim($answer,"\n");
            if($answer == ''){ $answer = null; }
            return $answer;
        }
    }
}
```

We then add a `getArguments()` method for retrieving the arguments, and move any existing user-input retrieval methods (previously in `coreCLI`) into this class.

UPDATING THE OUTPUT CLASS

We also need to update the `Output` class to handle both browser-based output and command-line output. Below is an example of how to handle CLI detection (`STDIN`) and color-coded messages.

Source: Output.php

src/Output.php

```php
<?php

/**
 * Core Framework - Output
 *
 * @license    MIT (https://mit-license.org/)
 * @author     Louis Ouellet
 */

// Declaring namespace
namespace LaswitchTech\Core;

// Import additionnal class into the global namespace
use Exception;

class Output {

    // Properties
    protected $Colors = [
        "default" => "3[39m",
        "black" => "3[30m",
        "red" => "3[31m",
        "green" => "3[32m",
        "yellow" => "3[33m",
        "blue" => "3[34m",
```

```
        "magenta" => "3[35m",
        "cyan" => "3[36m",
        "light-gray" => "3[37m",
        "dark-gray" => "3[90m",
        "light-red" => "3[91m",
        "light-green" => "3[92m",
        "light-yellow" => "3[93m",
        "light-blue" => "3[94m",
        "light-magenta" => "3[95m",
        "light-cyan" => "3[96m",
        "white" => "3[97m",
    ];

    /**
     * Constructor
     */
    public function __construct(){}

    /**
     * Output a string
     */
    public function print($string, $httpHeaders=array()) {

        // Check if the script is running in CLI mode
        if(defined('STDIN')){
            // Print to the console
            print_r($string . PHP_EOL);
        } else {
            // If not CLI, handle browser output and headers
            if (!headers_sent()) {
                header_remove('Set-Cookie');
                if (is_array($httpHeaders) && count($httpHeaders)) {
                    foreach ($httpHeaders as $httpHeader) {
                        header($httpHeader);
                    }
                }
                if(is_array($string) || is_object($string)){
                    $string = json_encode($string,
JSON_UNESCAPED_SLASHES | JSON_PRETTY_PRINT);
                }
                echo $string;
                exit;
            }
```

```
        }
    }

    /**
     * Output a string with color
     */
    public function set($string, $color = 'default'){
        if(defined('STDIN') && isset($this->Colors[$color])){
            return $this->Colors[$color] . $string .
$this->Colors['default'];
        } else {
            return $string;
        }
    }

    public function error($string) {
        $this->print($this->set($string, 'red'));
    }

    public function success($string) {
        $this->print($this->set($string, 'green'));
    }

    public function warning($string) {
        $this->print($this->set($string, 'yellow'));
    }

    public function info($string) {
        $this->print($this->set($string, 'cyan'));
    }

    /**
     * Output command-line help
     */
    public function help($string = null) {

        // Check if the script is running in CLI mode
        if(!defined('STDIN')){ return; }

        // Retrieve Global Variables
        global $REQUEST, $CONFIG;
```

```
        // Retrieve the arguments
        $arguments = $REQUEST->getArguments();

        // Initialize Variables
        $file = $arguments[0] ?? null;
        $command = $arguments[1] ?? null;
        $action = $arguments[2] ?? null;

        // If a string is provided, print it
        if($string){ $this->print($string); }

        // Attempt to locate a specific command
        if($command){
            $path = $CONFIG->root() . "/Command/" . ucfirst($command) .
"Command" . ".php";
            if(!is_file($path)){
                $path = $CONFIG->root() . "/lib/plugins/" .
ucfirst($command) . "/Command.php";
                if(!is_file($path)){
                    $command = null;
                }
            }
        }
        if($command){
            if(!class_exists(ucfirst($command) . "Command")){
                require_once $path;
                if(!class_exists(ucfirst($command) . "Command")){
                    $command = null;
                }
            }
        }

        if($command){
            // If command found, display usage for that command
            $class = ucfirst($command) . "Command";
            $class = new $class();

            if(!method_exists($class, $action . "Action")){
                $action = null;
            }

            $this->print("Usage: $file $command [action] [options]");
            $this->print("Available Actions:");
```

```
            foreach(get_class_methods($class) as $method){
                if(substr($method,-6) == 'Action'){
                    $this->print(" - " .
strtolower(str_replace('Action','',$method)));
                }
            }
        } else {
            // General usage help
            $this->print("Usage: $file [command] [action] [options]");
            $this->print("Available Commands:");

            // From /Command/
            foreach(scandir($CONFIG->root() . "/Command/") as $command){
                if(str_contains($command, 'Command.php')){
                    $this->print(" - " .
strtolower(str_replace('Command.php','',$command)));
                }
            }

            // From /lib/plugins/
            foreach(scandir($CONFIG->root() . "/lib/plugins/") as
$command){
                if(is_file($CONFIG->root() . "/lib/plugins/" . $command
. "/Command.php")){
                    $this->print(" - " . strtolower($command));
                }
            }
        }

        // Stop execution
        exit();
    }
}
```

IMPLEMENTING HELPERS

Helpers provide small, shareable methods. To achieve this, we first create a base `Helper` class

that can be extended if we need shared properties or methods in the future.

Source: Helper.php

src/Helper.php

```php
<?php

/**
 * Core Framework - Helper
 *
 * @license    MIT (https://mit-license.org/)
 * @author     Louis Ouellet
 */

// Declaring namespace
namespace LaswitchTech\Core;

class Helper {
    // Currently empty, but can house shared logic for helpers
}
```

Next, we implement the main `Helpers` class to dynamically load individual helper files from both your application and any plugins.

Source: Helpers.php

src/Helpers.php

```php
<?php

/**
 * Core Framework - Helpers
 *
 * @license    MIT (https://mit-license.org/)
 * @author     Louis Ouellet
 */
```

```php
// Declaring namespace
namespace LaswitchTech\Core;

// Import additionnal class into the global namespace
use Exception;

class Helpers {

    // Properties
    protected $Path = null;
    protected array $Helpers = []; // store them here

    public function __construct() {

        // Import Global Variables
        global $CONFIG;

        // Load application Helpers
        $this->Path = $CONFIG->root() . "/Helper";
        if(is_dir($this->Path)){
            foreach(scandir($this->Path) as $helper){
                if (preg_match('/(.+)Helper\.php$/i', $helper,
$matches)) {
                    require_once $this->Path . "/" . $helper;
                    $baseName = $matches[1];
                    $className = $baseName . 'Helper';
                    if (class_exists($className)) {
                        $this->Helpers[$baseName] = new $className();
                    }
                }
            }
        }

        // Load plugin Helpers
        $this->Path = $CONFIG->root() . "/lib/plugins";
        if(is_dir($this->Path)){
            foreach(scandir($this->Path) as $plugin){
                // Only load if not previously loaded
                if(!isset($this->Helpers[ucfirst($plugin)])){
                    $path = $this->Path . "/" . $plugin . "/Helper.php";
                    if(is_file($path)){
                        $baseName = ucfirst($plugin);
```

```
                        $className = $baseName . 'Helper';
                        // Attempt to load if the class exists
                        if (class_exists($className)) {
                            $this->Helpers[$baseName] = new
$className();
                        }
                    }
                }
            }
        }

    // Magic getters/setters
    public function __get($name) {
        return $this->Helpers[$name] ?? null;
    }

    public function __set($name, $value) {
        $this->Helpers[$name] = $value;
    }
}
```

IMPLEMENTING MODELS

Models behave much like Helpers, but differ in that they usually require a database connection. We store a reference to the global `$DATABASE` object in the base `Model` class.

Source: Modal.php

src/Modal.php

```
<?php

/**
 * Core Framework - Model
 *
 * @license    MIT (https://mit-license.org/)
```

```
 * @author     Louis Ouellet
 */

namespace LaswitchTech\Core;

use Exception;

class Model {

    // Properties
    protected $Database;

    public function __construct(){
        // Retrieve the global Database
        global $DATABASE;
        $this->Database = $DATABASE;
    }
}
```

Just like Helpers, the `Models` class dynamically loads all models from your `/Model` folder and plugin directories:

Source: Modals.php

src/Modals.php

```
<?php

/**
 * Core Framework - Models
 *
 * @license    MIT (https://mit-license.org/)
 * @author     Louis Ouellet
 */

namespace LaswitchTech\Core;

use Exception;
```

```
class Models {

    protected $Path = null;
    protected array $Models = [];

    public function __construct() {

        global $CONFIG;

        // Load application Models
        $this->Path = $CONFIG->root() . "/Model";
        if(is_dir($this->Path)){
            foreach(scandir($this->Path) as $model){
                if (preg_match('/(.+)Model\.php$/i', $model, $matches))
{
                    require_once $this->Path . "/" . $model;
                    $baseName = $matches[1];
                    $className = $baseName . 'Model';
                    if (class_exists($className)) {
                        $this->Models[$baseName] = new $className();
                    }
                }
            }
        }

        // Load plugin Models
        $this->Path = $CONFIG->root() . "/lib/plugins";
        if(is_dir($this->Path)){
            foreach(scandir($this->Path) as $plugin){
                if(!isset($this->Models[ucfirst($plugin)])){
                    $path = $this->Path . "/" . $plugin . "/Model.php";
                    if(is_file($path)){
                        $baseName = ucfirst($plugin);
                        $className = $baseName . 'Model';
                        if (class_exists($className)) {
                            $this->Models[$baseName] = new $className();
                        }
                    }
                }
            }
        }
    }
```

```
    public function __get($name) {
        return $this->Models[$name] ?? null;
    }

    public function __set($name, $value) {
        $this->Models[$name] = $value;
    }
}
```

IMPLEMENTING THE COMMAND-LINE MODULE

To implement a command-line interface (CLI), we'll create a template `Command` class for developers to extend. Then we'll add a `CLI` class that identifies and executes commands and actions.

BASE COMMAND CLASS

Source: Command.php

src/Command.php

```
<?php

/**
 * Core Framework - Command
 *
 * @license    MIT (https://mit-license.org/)
 * @author     Louis Ouellet
 */

namespace LaswitchTech\Core;

use Exception;
```

```
class Command {

    // Global Properties
    protected $Model;
    protected $Helper;
    protected $Output;
    protected $Request;

    public function __construct(){
        global $MODEL, $HELPER, $OUTPUT, $REQUEST;
        $this->Model = $MODEL;
        $this->Helper = $HELPER;
        $this->Output = $OUTPUT;
        $this->Request = $REQUEST;
    }

    /**
     * Magic Method to catch all undefined methods
     */
    public function __call($name, $arguments) {
        $this->Output->help();
    }
}
```

CLI CLASS

Source: CLI.php

src/CLI.php

```
<?php

/**
 * Core Framework - CLI
 *
 * @license    MIT (https://mit-license.org/)
 * @author     Louis Ouellet
```

```
 */

namespace LaswitchTech\Core;

use Exception;

class CLI {

    protected $Output;
    protected $Request;
    protected $Config;

    protected $Arguments;
    protected $Class;
    protected $Path;
    protected $Command;
    protected $Action;

    public function __construct(){
        global $OUTPUT, $REQUEST, $CONFIG;
        $this->Output = $OUTPUT;
        $this->Request = $REQUEST;
        $this->Config = $CONFIG;
        $this->Arguments = $this->Request->getArguments();
    }

    public function run(){
        // Parse Standard Input
        if(count($this->Arguments) > 0){
            // The first argument is the script name
            unset($this->Arguments[0]);

            if(count($this->Arguments) > 0){
                // Identify the Command
                $this->Command = ucfirst($this->Arguments[1] .
"Command");
                unset($this->Arguments[1]);

                // Check path in /Command/ and /lib/plugins/
                $this->Path = $this->Config->root() . "/Command/" .
$this->Command . ".php";
                if(!is_file($this->Path)){
```

```
                    $this->Path = $this->Config->root() .
"/lib/plugins/" . strtolower(str_replace('Command','',$this->Command)) .
"/Command.php";
                }

                if(is_file($this->Path)){
                    require_once $this->Path;
                    if(class_exists($this->Command)){
                        $this->Class = new $this->Command();

                        if(count($this->Arguments) > 0){
                            $this->Action = $this->Arguments[2] .
"Action";
                            unset($this->Arguments[2]);

                            if(method_exists($this->Class,
$this->Action)){
                                $this->Class->{$this->Action}();
                            } else {
                                $this->Output->help("The action " .
strtolower(str_replace('Action','',$this->Action)) . " is not
available");
                            }
                        } else {
                            $this->Output->help();
                        }
                    } else {
                        $this->Output->help("Could not initialize the
command " . strtolower(str_replace('Command','',$this->Command)));
                    }
                } else {
                    $this->Output->help("The command " .
strtolower(str_replace('Command','',$this->Command)) . " is not
available");
                }
            } else {
                $this->Output->help();
            }
        } else {
            $this->Output->help("Internal Error");
        }
    }
```

```
}
```

TESTING THE CODE: CREATING A "DEBUG" PLUGIN

Let's create a simple plugin named `debug` in `/lib/plugins/debug/Command.php`. This demonstrates how to define custom commands to be handled by our new CLI infrastructure:

Source: Command.php

lib/plugins/debug/Command.php

```php
<?php

/**
 * Core Framework - DebugCommand
 *
 * @license    MIT (https://mit-license.org/)
 * @author     Louis Ouellet
 */

use LaswitchTech\Core\Command;

class DebugCommand extends Command {

    public function __construct(){
        parent::__construct();
    }

    public function executeAction(){
        $this->Output->print("Debugging...");
    }
}
```

RUNNING THE CLI

Below is a sample script `debug.php` in a `tmp/` folder to test the CLI. Make it executable (`chmod +x tmp/debug.php`) and run it like so: `./tmp/debug.php debug execute`

Source: debug.php

tmp/debug.php

```
#!/usr/bin/env php
<?php

use LaswitchTech\Core\Bootstrap;
use LaswitchTech\Core\CLI;

require dirname(__DIR__) . "/vendor/autoload.php";

// Initiate Bootstrap
$BOOTSTRAP = new Bootstrap("CLI");

// Define EOL based on environment
$_EOL = defined("STDIN") ? PHP_EOL : "<br>";

// Display some debugging info
foreach(get_defined_vars() as $key => $value){
    if(substr($key,0,1) == "_" || in_array($key,["argv","argc"]))
continue;
    echo "Variable Name: " . $key . " = " . get_class($value) . $_EOL;
}

echo $_EOL . "Host: " . $REQUEST->getHostAddress() . $_EOL;
echo $_EOL . "Parameter: " . $REQUEST->getParams("GET","query") . $_EOL;

if(defined('STDIN') && !empty($argv)){
    echo $_EOL . "CLI: " . $_EOL;
    $CLI->run();
}
```

```php
echo $_EOL . "End of Script" . $_EOL;
```

EXAMPLE OUTPUT

When you run:

```
./tmp/debug.php debug execute
```

You should see something like:

```
Variable Name: BOOTSTRAP = LaswitchTech\Core\Bootstrap
Variable Name: REQUEST = LaswitchTech\Core\Request
Variable Name: CLI = LaswitchTech\Core\CLI
Variable Name: CONFIG = LaswitchTech\Core\Config
Variable Name: LOG = LaswitchTech\Core\Log
Variable Name: OUTPUT = LaswitchTech\Core\Output
Variable Name: NET = LaswitchTech\Core\Strap
Variable Name: HELPER = LaswitchTech\Core\Helpers
Variable Name: DATABASE = LaswitchTech\Core\Strap
Variable Name: MODEL = LaswitchTech\Core\Models
Variable Name: LOCALE = LaswitchTech\Core\Strap
Variable Name: SLS = LaswitchTech\Core\Strap
Variable Name: SMS = LaswitchTech\Core\Strap
Variable Name: SMTP = LaswitchTech\Core\Strap
Variable Name: IMAP = LaswitchTech\Core\Strap
Variable Name: INSTALLER = LaswitchTech\Core\Strap

Host: localhost

Parameter:

CLI:
Debugging...

End of Script
```

CONCLUSION

GITHUB REPOSITORY

GitHub Repository - v0.0.7

In this second part of our series, we expanded our modular PHP framework to support command-line operations, helper classes, and models. By creating a generic `CLI` class and a template `Command` class, we can easily extend the framework with new commands and actions. The addition of `Helpers` and `Models` provides a cleaner, more organized way to share code across your application—making it easier to maintain and extend functionality without duplicating code.

Our framework is starting to take shape with clear separation of concerns and the flexibility to add or remove features as needed. In upcoming parts, we will continue to refine our framework, possibly by integrating more robust logging, configuration management, and additional modules that can further streamline development.

RELATED ARTICLES

- Let's Talk - Building a Modular PHP Framework from Scratch
- Let's Talk - Building a Modular PHP Framework part 2
- Let's Talk - Building a Modular PHP Framework part 3

TAGSGENERALCORE-FRAMEWORK-ENFRAMEWORKCOREMODULARITYMODULARMODULELIGHTWEIGHTAPPLICATIONPHPCAKEPHPSYMFONYLEARNLEANINGMAINTAINABILITYMAINTAIN

- Twitter
- Facebook
- LinkedIn

- Reddit
- Telegram
- Email

View the discussion thread.

From:
https://laswitchtech.com/ - **LaswitchTech**

Permanent link:
https://laswitchtech.com/en/blog/2025/01/28/let-s-talk-building-a-modular-php-framework-part-2

Last update: **2026/02/16 13:20**