CB

General information

Inheritance	no
Child classes	no
Interfaces	ArrayAccess
Source	Framework/cb.php

Class CB is the main class that implements basic functionality of the framework:

- Catching and error handling.
- Class lazy loading.
- Application configuration.
- Registry design pattern implementation.
- Simplest means of logging and profiling.

Besides the above functional CB provides several additional methods that can be useful in development of any web application.

Public static methods

General methods

They may be used to get various information or perform frequent operations.

init()

```
public static self init()
```

Framework initialization. Returns object of class CB.

getInstance()

```
public static CB getInstance()
```

Returns instance of CB, or NULL, if the framework is not initialized (method `CB::init()` was not invoked).

getRoot()

```
public static string getRoot()
```

Returns full path to the site root directory (DOCUMENT ROOT). If the framework is not initialized the method returns NULL.

getSiteUniqueID()

```
public static string getSiteUniqueID()
```

Returns unique identifier of the application, which is defined as md5 from `$_SERVER['DOCUMENT_ROOT']`.

setOutput()

```
public static void setOutput(string $output)
```

$output	string	response body.

Sets the body of the server response, with all of the information is already ranked in the output buffer will be ignored.

getOutput()

```
public static string getOutput()
```

Returns previously defined body of the server response.

exe()

```
public static string exe(string $code, array $vars = null)
```

$code	string	inline PHP code to perform.
$vars	array	associative array of variables that will be extracted in PHP code.

Performs a built-in HTML PHP code and returns the result.

dir()

```
public static string dir(string $dir)
```

$dir	string	path to a site directory or its alias.

Returns the full path to a directory site. Method parameter can be: a full path to the directory, path to the directory relative to the root site and alias directory specified in the configuration file of the site.

url()

```
public static string url(string $url)
```

$url	string	URL of a site directory or its alias.

Returns the URL of a site directory, defined by a path relative to the root of the site or its alias.

log()

```
public static void log(mixed $data)
```

$data	mixed	any data to log.

Writes arbitrary data to a log file. In addition to the data passed to the method will be recorded in the log additional information about the environment of the script (IP address, ID session, URL request, the current timestamp, the contents of the session, cookies, etc.)

go()

```
public static void go(string $url, boolean $inNewWindow = false, boolean $immediately = true)
```

$url	string	URL to redirect.
$inNewWindow	boolean	determines whether or not a new browser window opens.
$immediately	boolean	determines whether the redirect was immediately done and the script was stopped.

Navigates to the specified address. If the second argument is TRUE, a new browser window will open. Otherwise, the redirect will be done in the current window. The third parameter allows you to set deferred redirect if it equals to FALSE. In this case, after calling this method the script does not stop and the redirect will be made after full execution of the script.

reload()

```
public static void reload(boolean $immediately = true)
```

$immediately	boolean	determines whether a restart happens immediately and the script is terminated.

Performs reload the page. If the parameter is FALSE, the script is not terminated as a result of the method call and reload occurs after the full script execution. Otherwise, the script aborts your work and reboot is performed immediately.

delegate()

```
public static mixed delegate(mixed $callback, mixed $arg1, mixed $arg2, ...)
```

$callback	mixed	delegate's string, delegate's object or any callable object.
$arg1, **$arg2**, ...	mixed	Callback arguments.

Invokes callback function as delegate.

Profiling methods

Allow to measure the time of executing arbitrary code sections, as well as receive information on the memory consumed during the execution of the script.

pStart()

```
public static void pStart(string $key)
```

$key	string	unique identifier of a profiled code block.

Remember the current time value for the profiled code block.

pStop()

```
public static float pStop(string $key)
```

$key	string	unique identifier of a profiled code block.

Returns execution time of the code block by its identifier. If no code block is associated with the passed identifier the method returns FALSE.

getExecutionTime()

```
public static float getExecutionTime()
```

Returns script execution time in seconds. The beginning of script execution is the first call of method **init**.

getRequestTime()

```
public static float getRequestTime()
```

Returns request time in seconds.

getMemoryUsage()

```
public static integer getMemoryUsage()
```

Returns the number of the script's memory in bytes.

getPeakMemoryUsage()

```
public static integer getPeakMemoryUsage()
```

Returns the peak of memory, in bytes, of the script.

Registry methods

The combination of these methods is an implementation of Registry design pattern - global pool of public objects of different types.

all()

```
public static array all()
```

Returns associative array of global objects.

get()

```
public static mixed get(string $key)
```

$key	string	a key, which associated with some global object.

Returns previously stored global object, or NULL if an object with such key does not exist.

set()

```
public static void set(string $key, mixed $value)
```

$key	string	a key, which associated with some global object.
$value	mixed	value of a global object.

Binds a value to the specified key and stores it in the storage of global objects.

has()

```
public static boolean has(string $key)
```

$key	string	a key, which associated with some global object.

Returns TRUE, if the specified key exists for the global object, and FALSE otherwise.

remove()

```
public static void remove(string $key)
```

$key	string	a key, which associated with some global object.

Removes global object from the storage by its key.

Error handling methods

These methods allow you to catch all kinds of errors and exceptions as well as to control the framework's reaction to these errors.

errorHandling()

```
public static void errorHandling(boolean $enable = true, integer $errorLevel = null)
```

$enable	boolean	a sign of the exception (error) handling mode.
$errorLevel	integer	sensitivity level to errors.

Turns on or off error handling, and sets the sensitivity to errors. If the first parameter is TRUE, the error handling mode will be enabled in which any error or exception in the script will be catched and processed by framework. The second option allows you to set the sensitivity to certain types of errors. If this parameter is not specified, it will set the sensitivity level specified in php.ini

isErrorHandlingEnabled()

```
public static boolean isErrorHandlingEnabled()
```

Returns TRUE, if the error handling mode is enabled and FALSE otherwise.

error()

```
public static string error(string|object $class, string $token, mixed $var1, mixed $var2, ...)
```

$class	string, object	class containing a token error.
$token	string	error token, an error template which is defined in some class as a constant.
$var1, **$var2**, ...	mixed	parameters (variables) of the error template.

Generates the error message given by some error token (pattern). Parameters (variables) within the error templates are specified using a special notation: [{var}]. Replacement of template variables to their actual values occurs from left to right.

There are three main cases depending on the passed parameters:

- Error token is a regular string which is not a constant of some class. In this case the first parameter equals an empty string or FALSE. For example:

```
// Displays 'Simple error template: 1, 2';
echo CB::error(false, 'Simple error template: [{var}], [{var}]', 1, 2);
```

- Error token is a constant of some class. And that class is set as an object. For example:

```
class A
{
  const ERR_1 = '[{var}] error template[{var}]';
}

$a = new A;

// Displays 'Test error template! (Token: A::ERR_1)'
echo CB::error($a, 'ERR_1', 'Test', '!');
```

- Error token is a constant of some class. Instead of a class object is passed the name of the class. Example:

```
class A
```

```
{
  const ERR_1 = '[{var}] error template[{var}]';
}

$a = new A;

// Displays 'Test error template! (Token: A::ERR_1)'
echo CB::error('A', 'ERR_1', 'Test', '!');

// It is also possible a more compact version of the record
// (with the same result):
echo CB::error('A::ERR_1', 'Test', '!');
```

exception()

```
public static void exception(Exception $e)
```

$e	Exception	object of exception.

Terminates the script and displays, in the case of debug mode is enabled, information of the exception which passed as a method parameter. If debug mode is turned off the method will print a notice about the error.

analyzeException()

```
public static array analyzeException(Exception $e)
```

$e	Exception	object of exception.

Returns detailed information about the exception as an associative array.

ecode()

```
public static string ecode(string $code)
```

$code	string	some PHP code.

Saves a string PHP code passed to a method for analysis of errors that can occur when you run this code, the operator eval. The method returns the same value as that takes as an argument.

During the development process there are situations when you need to execute some php-code in the script. This often used construction eval. However, debugging errors is difficult in this case. To improve the process of debugging the code executed by eval and designed this method. All you need is to combine the call to the operator ecode eval. Example:

```
eval(CB::ecode("echo 'Hello World!';"));
```

fatal()

```
public static void fatal()
```

Used to detect fatal errors or parsing errors. The method is called automatically when the script is over.

Public non-static methods

Configuration methods

These methods are designed for loading configuration files for the application, as well as for access to configuration data.

setConfig()

```
public self|array setConfig(string|array $data, string $section = null, boolean $replace = false)
```

$data	string, array	path to the configuration file, or an associative array of configuration data.
$section	string	the configuration section name.
$replace	boolean	determines whether to replace existing data with new configuration.

Loads the configuration data from a file or an array. The second method parameter determines the configuration section in which the configuration data will be loaded. If you want to overwrite the old data with new data, **$replace** should be TRUE.

Configuration files must be INI or PHP files. The typical structure of the PHP configuration file:

```
<?php

return ['var1' => 'val1',
        'var2' => 'val2',
        ...,
        'section1' => ['var1' => 'val1'
                       'var2' => 'val2',
                       ...],
        'section2' => ['var1' => 'val1'
                       'var2' => 'val2',
                       ...],
        ...];
```

The typical INI file structure:

```
var1 = "val1"
var2 = "val2"
...

[section1]
var1 = "val1"
var2 = "val2"
...

[section2]
var1 = "val1"
var2 = "val2"
...
```

The below examples illustrate the usage of the method:

```
$cb = \CB::getInstance();
// loads config data from the file.
$cb->setConfig('/path/to/config/file.php');
// loads config data to the section "foo".
$cb->setConfig(['var1' => 'val1', 'var2' => 'val2'], 'foo');
// replace section "foo" new data from the config file.
$cb->setConfig('/path/to/config/file.ini', 'foo', true);
```

getConfig()

```
public mixed getConfig(string $section = null)
```

$section	string	the configuration section name.

If **$section** is NULL the method returns the all configuration data, otherwise the content of the given configuration section will be returned. In the

case if none section with the defined name exists the method returns NULL.

Let's say we have the following configuration INI file:

```
debugging = 1
logging   = 1

[cache]
directory     = "cache"
gcProbability = 33.333

[foo]
var1 = "val1"
var2 = "val2"
```

The same file but on PHP:

```
<?php return ['debugging' => 1,
              'logging'   => 1,
              'cache'     => ['directory'     => 'cache',
                              'gcProbability' => 33.333],
              'foo'       => ['var1' => 'val1',
                              'var2' => 'val2']];
```

The below examples illustrate the usage of the method:

```
$cb = \CB::getInstance();
$foo = $cb->getConfig('foo');  // $foo contains the content of "foo" section.
$foo = $cb->getConfig('test'); // $foo contains NULL.
$foo = $cb->getConfig();       // $foo contains all configuration data.
```

offsetSet()

```
public void offsetSet(mixed $var, mixed $value)
```

$var	mixed	name of a configuration variable.
$value	mixed	value of a configuration variable.

Adds a new configuration variable or modifies the existing one. The method is part of the interface **ArrayAccess**. Example:

```
// Gets an instance of ClickBlocks
$a = CB::getInstance();

// Add new variable "foo"
$a['foo'] = 'test';
```

offsetGet()

```
public mixed offsetGet(mixed $var)
```

$var	mixed	name of a configuration variable.

Returns the value of the configuration variable by its name. If a variable with this name does not exists, the method returns NULL. The method is part of the interface **ArrayAccess**. Example:

```
// Gets an instance of CB
$a = CB::getInstance();

// Gets value of variable "foo"
```

```
echo $a['foo'];
```

offsetExists()

```
public boolean offsetExists(mixed $var)
```

$var	mixed	name of a configuration variable.

Returns TRUE, if the configuration variable exists and is not NULL, and FALSE otherwise. The method is part of the interface **ArrayAccess**. Example:

```
// Gets an instance of CB
$a = CB::getInstance();

// Checks whether variable "foo" exists or not.
var_dump(isset($a['foo']));
```

offsetUnset()

```
public void offsetUnset(mixed $var)
```

$var	mixed	name of a configuration variable.

Removes a configuration variable by its name. The method is part of the interface **ArrayAccess**. Example:

```
// Gets an instance of CB
$a = CB::getInstance();

// Removes variable "foo"
unset($a['foo']);
```

Class loader methods

Used to control the operation of the class loader, as well as to explicitly load classes.

createClassMap()

```
public integer createClassMap()
```

Searches classes according to the configuration settings and creates the class map file. The method returns the number of the all found classes.

setClassMap()

```
public void setClassMap(array $classes, string $classmap = null)
```

$classes	array	the array of paths to the files with classes.
$classmap	string	the path to the class map file.

Creates new class map or replaces old one.

getClassMap()

```
public array getClassMap()
```

Returns the class map array which contains class names as its keys and paths to class files as its values. If the class map is not set the method returns empty array.

loadClass()

```
public boolean loadClass(string $class)
```

$class	string	the name of the class to be loaded.

Searches a single class and includes it to the script. Returns FALSE if the given class does not exist and TRUE otherwise.

Additional methods

Allow to obtain objects of the most frequently used classes.

setCache()

```
public void setCache(ClickBlocks\Cache\Cache $cache)
```

$cache	ClickBlocks\Cache\Cache	cache object.

Sets the default cache object. This cache object will be used by default for all cache operations in the framework.

getCache()

```
public ClickBlocks\Cache\Cache getCache()
```

Returns the default cache object. If the default cache object is not set by the method **CB::setCache()**, it will be created according to the configuration settings.

getRequest()

```
public ClickBlocks\Net\Request getRequest()
```

Returns an object of class **ClickBlocks\Net\Request**, which contains information about the current HTTP-request.

getResponse()

```
public ClickBlocks\Net\Response getResponse()
```

Returns an object of class **ClickBlocks\Net\Response**, which contains information about the server response on the current HTTP-request.

getRouter()

```
public ClickBlocks\Net\Router getRouter()
```

Returns an object of the router class.