Simple MVC Framework - Install guide

Yoann Mikami

January 1, 2013

About this package

This package contains the files needed to build a web application using an MVC[1] oriented architecture.

This PDF describes the steps needed to setup the initial environment needed to develop a new web application. This can also be used as a reference for existing sites as well.

Contents

[1] (Model-View-Controller)

1 Installation

1.1 Requirements

OS

Linux (untested on Windows OSes)

Software

- PHP > 5.2
- Apache 2.x
- MySQL 5.x or MongoDB 1.x

1.2 Dependencies

Below are the 3rd party packages/frameworks used by this framework. They should all be supplied under the *opt/* directory so nothing is needed.

[**WideImage** http://wideimage.sourceforge.net/]
OO Image manipulation framework. Used by `Utils_ResourceLoader_Image`. Licence : GNU LGPL 2.1

[**savant3** http://phpsavant.com]
Savant3 templating engine - not maintained anymore. Licence: GNU LGPL

[**spyc** https://github.com/mustangostang/spyc/]
Simple PHP YAML Class : YAML parser. Licence : MIT

[**Zend** http://framework.zend.com]
Zend Framework (partial, 1.12). Used packages :

- Cache
- Db
- Json
- Loader
- Locale
- Log
- Registry
- Session
- Translate

1.3 Recommended structure for a web application

The developer is free to choose where everything is stored under the server docroot. The following is the recommendation (folder name can be different for each module) :

framework
: This framework. May be shared

my_site
: The web application specific files (models, controllers, views, layouts, scripts, css. . .)

my_site_resources
: Files handled by the server, user supplied resources (application logs, cached files, uploaded files. . .)

docroot
: The entry point to web applications

These four subparts can either be located at a single location (not recommended), or split (recommended). Split rationales :

framework
: This framework serves as the base for a web application, and thus can be used by many different ones.
Putting this framework under its specific location allows to share the same code among several web applications installed on the same server.
See the ➢*Developer guide* for the framework structure.

my_site
: Separating the site core files (managed by the webmaster/developer) from the server/user resources

resources
: Eases maintenance; the site files can be packaged into a *tarball* and extracted to the proper location, without risking to overwrite files that are not part of it. Logs and cached files can be preserved easily as well. server resources folders usually have special folder permissions and thus is more easily managed when it is separated.

docroot
: For security purposes, docroot should have its own directory with a small bootstrap needed to start the application.
The intialization class `Init` (in this framework) can be called by the bootstrap by pointing to its location (outside docroot).
This setup assumes a single docroot is shared among all web applications installed on the server. This can be changed by creating an additional folder "*my_site*" which would contain folders "*my_site*", *resources* and *docroot*.

This framework uses "application namespace concept" to separate the different web application contents. (This concept is explained in more details later) An application namespace allows inter-related applications to share the same workspace. A typical use of namespace would be a web site with a front-end (typically read only) and a back-end (e.g. the admin interface). Both have shared resources, shared assets, yet some differences may be needed at some specific locations.

> ⓘ Namespacing should not be used for two UNRELATED web applications, as most resources are *shared*, ie. both applications can access each other's assets, which is not a good idea in general in this use case. A different folder (e.g. "*my_site2*") should be created instead.

1.4 Creating the application structure

A web application should have the following structure :

```
"site root dir"
├── config  The web application config file (Recommended : chmod 0775)
├── controllers  location for controllers used by the web application
├── langs  language files (gettext)
├── layouts  location for template files
├── libs  PHP classes
│   ├── opt  Application specific 3rd party tools
│   ├── plugins  init, view, resource plugins
│   │   ├── Init  Init plugins folder
│   │   ├── View  View plugins folder
│   │   └── Resource  Resource plugins folder
│   └── savant3
│       └── Savant3
│           └── resources  Application specific Savant3 template filters and plugins folder
├── models  location for models used by the web application
├── skins  Site resources : graphics, CSS, javascript
└── views  location for view files used by the web application
```

1.5 Creating the server resources structure

When separated from the site structure, the server resources should be structured like the following :

```
"Resources root dir"
├── cache  cached files (if caching is enabled)
│   (Recommended : chmod 2775, chown www-data:www-data)
├── logs  The web application logs
│   (Recommended : chmod 2775, chown www-data:www-data)
└── upload  user uploaded files
    (Recommended : chmod 0775)
```

Server resources and application structures can be merged if there is no need for separating them. If *resources* folder is external to the "*site root dir*", the latter should have a symlink pointing at the former :

```
"site root dir"
├── (...)
├── resources → [ "resources root dir" ]
└── (...)
```

It is also possible to mix *symlink* with physical folders inside a *resources* folder inside "*site root dir*" :

```
"site root dir"
├── (...)
└── resources  "resources root dir"
    ├── cache   Cache folder (physical)
    ├── logs   Log folder (physical)
    └── upload → [ "path to external upload folder" ]
```

2 Setup application namespace(s)

2.1 About namespaces

Through this framework, every web application have at least one namespace : the application itself. By default, a namespace is defined as the server name the application runs onto.
For instance, the default application namespace for `http://www.my_site.com/` would be www.my_site.com. Namespaces are heavily used in the framework;

Namespaces allow a logical separation of web applications while sharing physical resources. How an application is namespaced and how many it has has to identified by the developer.

2.1.1 Use cases

- A typical 2 namespaces web application is one with front/back ends. Thus, `http://www.my_site.com` would be the front-end, and `http://admin.my_site.com` the back-end. However, the code is shared for both of them, unless one overrides the other.

  ⓘ Apache mod_rewrite is required to allow namespacing to work.

- An intranet must be accessed through different credentials depending on the user role :
  - Recruiter, Interviewer, Administrator
  Each of these could yield to 3 application namespaces, such as recruiter.my_site, itw.my_site , admin.my_site. Each namespace would then provide different functionalities while keeping a global layout, sharing common features...
  While this could be used to apply simple permission control system, application namespacing does not replace a user permissions engine such as ACL[2] or Role Based Access Control
  .

[2] Access Control List

- An application is deployment in a development server, staging, production server that all require different settings. The config file uses application namespaces so each can use common settings while overridding whatever's needed. In such a case, the application would exactly be the same for all namespaces, which would be used solely for configuration.
- A web site has a PC and mobile version : both share the same resources and text, while the layout may differ to address smaller screen sizes.
  Two namespaces, mobile and pc can achieve this : page layout can be customized, while the content is still shared.

2.1.2 Namespace overridding

Application namespaces rely on inheritance.

To allow different namespaces to share common resources, one must inherit another. For case 1 above, admin namespace would thus inherit the front namespace. The other way round would not work as the former probably has more features than the latter, and inheritance allows to override the parent's assets or add your own, but not disable values set by the parent.
For case 4, mobile would inherit pc.

ⓘ The top application namespace should *always* inherit "default" in the configuration file.

When two or more namespaces are defined, the following process is done when accessing any namespaced resource. Knowing this allows to determine which namespace should inherit which :

Three namespaces are defined (excl. "default") :

```
baz --- overrides ---> bar --- overrides ---> foo --- overrides ---> default
```

When requesting resource `foobar` :

- Is foobar resource available under namespace baz ? If YES, then use it $\Longrightarrow$ [END]
- Is foobar resource available under namespace bar ? If YES, then use it $\Longrightarrow$ [END]
- Is foobar resource available under namespace foo ? If YES, then use it $\Longrightarrow$ [END]
- Resource not found $\Longrightarrow$ [END]

ⓘ The "default" namespace is not used during the process.
Exception: skin files also look for the "default" namespace as well. Non-namespace specific resources can be placed under it. It is only used to define top-level directories for the various resources the application can access. Under each of these, namespaces should be defined.

2.2 Define namespaces in the web application

For each namespace the application is expected to have (more can be added anytime), a folder named after the namespace should be created in the following folders :

```
Site root dir
└─config
  ├─controllers
  │ └─"namespace"
  ├─langs
  │ └─"namespace"
  ├─layouts
  │ └─"namespace"
  ├─libs
  ├─models
  │ └─"namespace"
  ├─skins
  │ ├─default
  │ └─"namespace"
  └─views
    └─"namespace"
```

- *config/config.yaml* contains the configuration for all namespaces in a single file and thus is not namespaced.
- *libs* is also shared among all namespaces.

Folders inside resources are not namespaced using folders :

- Under *logs/*, *cache/* generated files are prefixed with the application namespace.
- *upload* is not namespaced at all, and all user uploaded files are common to all namespaces. For namespace specific images. . . the application skins folder should be used instead.

3 Configuring the web application

The configuration file (a sample is available in this framework's *config-default* folder) *config.yaml* should be created under the "*site root dir*" *config/* directory. the namespace "default" is required, thus should always be defined; each namespace the web application uses should then have a top definition the same way "default" is defined in the sample *config.yaml*. When a namespace inherits (or overrides) another, the `overrides:` "*parent*" config definition should be used. Each namespace configuration definition only need to (re)define configuration properties that are different (or missing) from its parent. Refer to the ➢*Developer guide* for the configuration parameters.

4 Creating the bootstrap file

Each namespace should have its own bootstrap file located under the docroot folder for the application. *.htaccess* (or apache vhost config) may be used to dispatch based on specific conditions which bootstrap file to load. The bootstrap could by in any language; however, a PHP one is recommended to allow easier interaction with this framework.
The bootstrap only needs to do one action : call this framework's entry point, `Init::main()`.
It also needs to define some constants the framework assumes to be defined before init is started :

`DATA_DIR` [Required]
Absolute path to the $\boxed{\text{“\textit{site root dir}”}}$

`LIB_DIR` [Required]
Absolute path to the $\boxed{\text{“\textit{framework root dir}”}}$

`BOOTSTRAP_FILE` [Required]
Filename of the bootstrap file called; namely, the file being processed. Typically **`basename`** `(__FILE__)` when called from within the bootstrap file itself.

`NAMESPACE` [Optional]
Defaults to `$_SERVER['SERVER_NAME']` if not defined : namespace to use.
Namespace is mapped to the same record in the config file, as well as namespace folders inside each $\boxed{\text{“\textit{site root dir}”}}$ folder.

ⓘ '.' and '-' characters are transformed into '_' : thus, namespace defined in *config.yaml* should be named accordingly, e.g. www.mysite.com shall become www_mysite_com.
'-' is transformed into '_' for namespace folders, and the latter should thus be named accordingly, e.g. www.mysite-admin.com shall become www.mysite_admin.com

$\Longrightarrow$ This framework's docroot folder contains a sample PHP bootstrap named *namespace.php*.

5 Creating the .htaccess file

Depending on where the webroot points at (application's `docroot` directly or somewhere above, e.g. httpdocs), the content will be different; however, here are a few notes that should be applicable on most cases :

- Use mod_rewrite to reroute ALL requests targeting a specific server / domain to the php newly created bootstrap file. Different namespaces should be routed to separate bootstrap s (though not enforced).

- If $\boxed{\text{“\textit{site root dir}”}}$ is under `docroot`, disallow access to it : only the bootstrap can access it through PHP **`include`**s.

- The *.htaccess* file can be either in the same folder as the bootstrap files, or somewhere else. the `RewriteRule` should match the chosen organization accordingly.

6 Application ready. good luck!

Once the system is setup and the basic installation is done, it's time to add content to the application. ➢*Developer guide* and this framework *PHPDoc* generated documentation should contain everything needed to start build the application.

[END OF DOCUMENT]