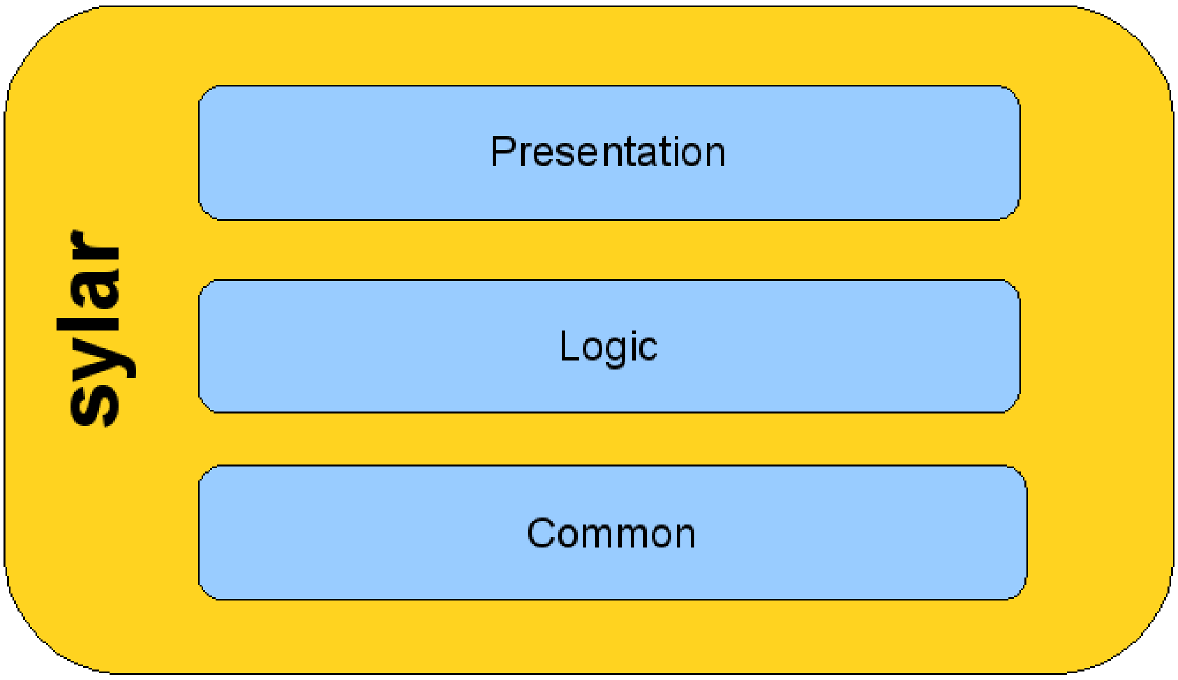
Sylar Deploy Guide Line [draft #2]

Sylar is an Open Source project with the aim of developing a framework to build PHP web applications. According to the Open Source philosophy all useful information are freely available (https://launchpad.net/sylar/) and the same development of the project is allowed to everyone based on ability and will.

Pattern

The Sylar project is conceived similarly to the MVC for web application like:

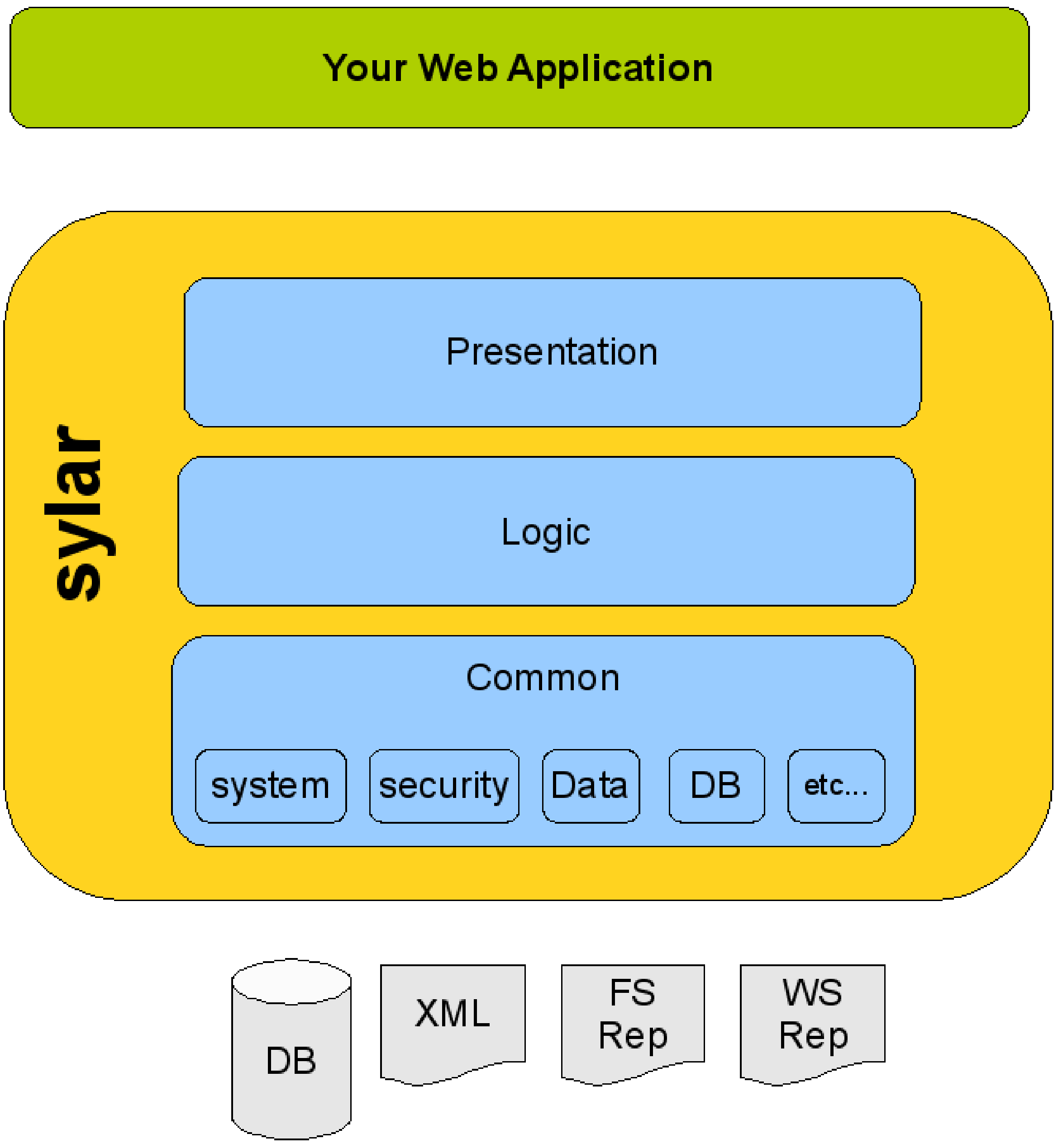


The framework has been created to keep constantly separated the access levels to the data, the logic and the user front-end.

The same approach is recommended for the development of your Sylar-based application.

Example of Sylar application

The typical *scenario* of web application created by means of Sylar is the follow:



The main characteristic is the clear separation between your web application and the data (or information) which could be supplied from several sources and could change with time.

Once the data is obtained, the task of   logic   will be to manipulate and to prepare them by its classes for further presentation to the user. Also this last point will be leave to the logic, in particular the so-called   presentation classes   will format the output data in a web page, in a mail, and so on.

Coding Rules

In the following, several fundamental rules to develop the Sylar project will be given. These rules define some common standards which enable an easier and faster development of the project with a code clear and easy understandable to all.

Source code documentation

A code is good if its documentation is good. Keeping in mind this idea, the first rule for developers is that code sources without a good documentation will be not considered and will be not inserted in the project.
The documentation must be written according to the PhpDoc standard that is similar to the JavaDoc one.
The easiest way to learn how to write a good documentation is to look at the documentation of existing Sylar files.
Some popular advanced development IDEs like Eclipse (free available) or Zend Studio could be very useful both to write php code and to write a good documentation.
Templates of documentation for different objects (classes, files, methods, and so on) will be given hereinafter.

<u>The first letter of all variables is lower case.</u>

Variable

The first letter of all variables is lower case.

A good rule (but not mandatory) is let primitive variables beginning with a letter that suggests their typology as in the example below:

```
      $iNum = 100;                        // integer
      $sName = "some text";               // string
      $bIsSystemUp = true;                // boolean
      $aDati = array();                   // array
      $oDbDriver = new Db();              // Object
      $fLong = 10.56;                     // floating point aka double
      $rResult = some_func_();            // resource
```

A good documentation of all the variables is a demand to make the code and its logic clear, and to generate an exhaustive documentation.

There should two types of comment in your code. Those useful to explain the code will be inserted as follows:

```
// Temp Var with name of user
$sNameUser = "Frank";
```

whereas those aimed at the documentation will be generated by means the following syntax:

```
/**
 * Email of user
 * @var string contains the user's email
 */
$sUserEmail = "pippo@pluto.com";
```

Classpath

The framework will include the classes stored in the root of Sylar packages in your filesystem as defined from the classpath in the Sylar configuration file:

sylar/settings/sylar.php

with the constant:

`SYLAR_CLASSES_ROOT_FS`

Your web application will have its own configuration file:

exampleApp/config/appConfig.php

and its own classpath saved in:

`SYLAR_APPLICATION_CLASSES_ROOT_FS`

Package

The name of a package is composed only from lower case letters.

Sylar organizes the code using the so-called packages, although they are not implemented till to the 5.3 version of PHP. A package is nothing else but a subdirectory of classpath housing classes and/or other packages.

An example of a typical Sylar package is:

`sylar.common.system`

and it will correspond to the directory:

SylarClasspath/common/system/

If you omit the   sylar.   prefix the framework will point to the classpath of your application. On the other hand, it will be possible to point to a particular package of your web application replacing   *sylar.*   with *app.* .

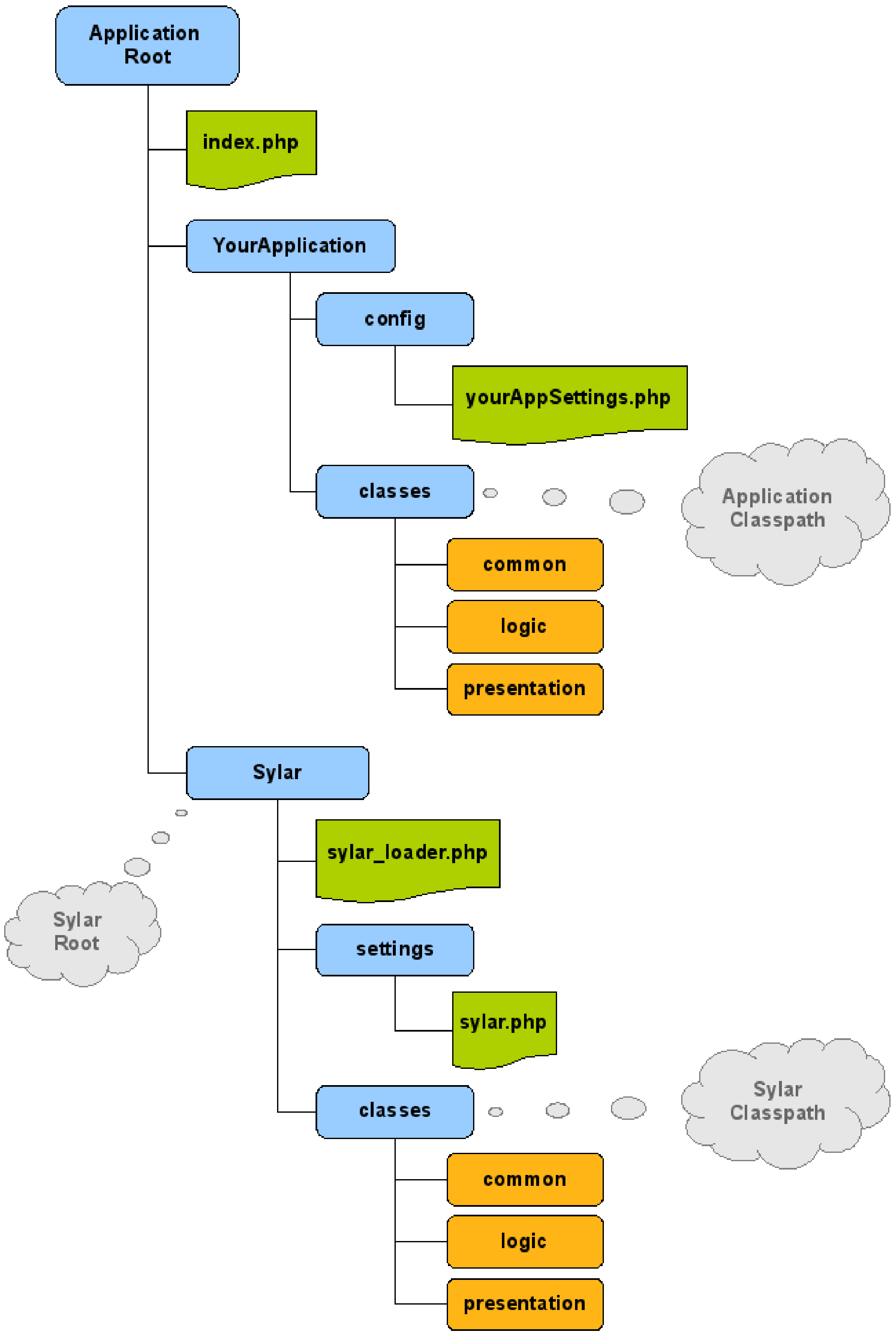
For example, the package:

`app.common.system.users`

will point to the following directory of the filesystem:

SylarClasspath/common/system/users/

Typically, a web application based on Sylar will have the filesystem structure showed in the following figure.


Application Root
index.php
YourApplication
config
yourAppSettings.php
classes
Application Classpath
common
logic
presentation
Sylar
Sylar Root
sylar_loader.php
settings
sylar.php
classes
Sylar Classpath
common
logic
presentation

Class

The first letter of all classes is upper case.

This simple rule makes possible to recognize immediately a class from a package or from a method which will begin with a lower case letter. To be more precise, it will be a good rule to start all meaningful word composing a class name with an upper case letter, as in example below:

```
Sylar_SimpleTableHeader
```

This let you easy understand what the class does.
All classes in Sylar will begin with the prefix *Sylar_* followed from a meaningful class name corresponding to the file name in the filesystem. Normally each class file contains only one class, or the main class that gives its name to the file.
In our example, the file will be:

SimpleTableHeader.php

> ***N.B.*** In the same file will be possible to insert more classes only if the guest class is used only and solely by the host class that gives its name to the file.

To use a class in the framework, you have import it using its package identifier. An example of Sylar class is:

```
sylar.common.data.DataContainer
```

while a simple example of application class will be:

```
app.giano.presentation.FormatLogList
```

An example of a good documentation will be:

```
/**
 * Configuration Box
 * A class that conteins all Sylar Configuration values read from
 * config and settings file.
 *
 * @package Sylar
 * @version 1.0
 * @since 16/feb/08
 * @author Gianluca Giusti [brdp] <g.giusti@giano-solutions.com>
 * @copyright Sylar Development Team
 */
```

Function and method

The first letter of all methods is lower case.

The name of a method begins with a lower case letter but the following principal words composing it will be upper case.

For example:

```
getColumnsList()
setTableName($sName)
isConnectionOpened($oDb)
```

In this way they will be easy identified from the classes. So, to call the static method *isSylarInDebugMode* belonging to the class *Sylar_ConfigBox* it will be use the following syntax:

```
Sylar_ConfigBox->isSylarInDebugMode()
```

with no chance to mistake.

The right documentation will be:

```
/**
 * Db type name
 * returns the name of db used, like mysql, oracle, ecc... The code must
 * be one of directory in the db package, for example:
 * db.mysql.MysqlDriver =>    name is "mysql"
 * db.oracle.OracleDriver     =>    name is "oracle"
 *
 * The name is also used tu include the class file
 *
 * @since 16/feb/08
 * @author Gianluca Giusti [brdp] <g.giusti@giano-solutions.com>
 *
 * @see SYLAR_USED_DB
 *
 * @todo to be done
 *
 * @return string The code of DbTyle like mysql, oracle
 * @param string $text text to parse
 * @param int iTotNodes number of node to parse
 */
```

Files and filenames

The file name of a file containing a class will be the name of the class without the prefix *Sylar_* as above clarified. It is very important to keep in mind that PHP is case sensitive, so pay attention.

Example of a file documentation:

```
/*
 * This file is part of Sylar.
 *
 * Foobar is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Foobar is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Foobar.  If not, see <http://www.gnu.org/licenses/>.
 *
 * @copyright Copyright Sylar Development Team
 * @license http://www.gnu.org/licenses/ GNU Public License V2.0
 * @see https://launchpad.net/sylar/
 * @see http://www.giano-solutions.com
 */
```

This sort of preamble is mandatory for all files. If the file contains more classes, a further documentation needs as for configuration files in which you have to insert a description like:

```
/**

 * Sylar - Framework Base Settings
 * Contains framework settings, constants and init instructions
 *
 * @package Sylar
 * @version 1.0
 * @since 02-2008
 * @author Gianluca Giusti [brdp] <g.giusti@giano-solutions.com>
 * @copyright Sylar Development Team
 */
```

Sylar Import

Sylar imports the files that it needs from the sylar or the application classpath using a function of the framework.

The syntax is quite simple:

```
import('sylar.common.sylar.db.mysql.MysqlDriver');   // import sylar class
import('app.giano.presentation.FormatLogList');      // import application class
import('giano.presentation.FormatLogList');          // import application class
```

If you don t insert the prefix *sylar.* or *app.* , the frame work will import the classes from the application classpath.

It is a good rule to import all the requested classes at the beginning of the file before everything else.

An example of class follows:

```php
<?php
/*
 * This file is part of Sylar.
 *
 * Foobar is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * Foobar is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with Foobar.  If not, see <http://www.gnu.org/licenses/>.
 *
 * @copyright Copyright Sylar Development Team
 * @license http://www.gnu.org/licenses/ GNU Public License V2.0
 * @see https://launchpad.net/sylar/
 * @see http://www.giano-solutions.com
 */

import('sylar.common.db.DataBaseDriver');
import('sylar.common.system.Logger');
import('sylar.common.system.ExceptionInSylar');


/**
 * Example Class
 *
 * A class designed as example in sylar
 * It's onlly an example
 *
 * @package Sylar
 * @version 1.0
 * @since 16/feb/08
 * @author Gianluca Giusti [brdp] <g.giusti@giano-solutions.com>
 * @copyright Sylar Development Team
 */
class Sylar_ExampleClass{
      private $sTitle = “name”;

      function __construct(){
            # nothing to do
      }
      function __destruct() {
            # nothing to do
      }

      public function setTableTitle($sTitle){
            $this->title = $sTitle;
      }
      public function getTableTitle(){
            return $this->title;
      }
}
?>
```

Project management

Presently, the Sylar project is developed and managed on the Launchpad.net platform. The reference web address is the home page of the project itself on this platform and all the documentation, bug management, FAQ, and so on are all available in Launchpad too.

Version control

Sylar uses bazaar to right update and handle the version number. This software is perfectly integrated into the Launchpad.net platform.