Contents

[Structure of the Module Installer 2](#_Toc370743265)

[ZEND Module 6](#_Toc370743265)

[Dependency Configuration 11](#_Toc370743266)

[Hooks Configuration 1](#_Toc370743267)2

[ACL Configuration 1](#_Toc370743267)3

[Settings Configuration 1](#_Toc370743267)4

[CUSTOM Modules 17](#_Toc370743268)

Structure of the Module Installer**:**

Modules can be developed in broadly two methodologies:

1. Using Zend framework - Using Zend Framework 2
2. Without using the framework

Thus the Modules, depending on their type, will be falling under

***/interface/modules/zend\_modules****and*

***/interface/modules/custom\_modules***

The `Installer` folder under the `interface\modules\zend\_modules\module` folder contains the installation script for the modules. Running the module installer will show the list of installed and new modules from which you can register, install and activate the modules. The below link shows the help for configuring Zend type modules

/interface/modules/zend\_modules – For managing all the zend based module code (Zend Framework2)

<http://packages.zendframework.com/docs/latest/manual/en/modules/zend.view.quick-start.html#zend-view-quick-start-usage-config>

Folder Definitions are detailed as below:

**/**

* composer.json
* inti\_autoloader.php
* LICENSE.txt
* index.php

- modified from default zend code to make the session and global variables of OpenEMR available to the modules inside the modules folder

**/config/**

* application.config.php

This will contain an array of configurations for each of the modules that are installed in the system. The edits in this file will be mostly to add the new module name to the modules array.

Eg: 'modules' =>array(

'Application',

'ModuleName',-add your module name here

),

**/config/autoload/**

This folder will actually contain the database configuration. This is of limited use since we are avoiding this to use the default OpenEMR libraries for db access and updates. However the configurations in these files will be autoloader based on the site configuration of the logged in user

* global.php

Default db configurations except the credentials

* local.php

Contains the db user credentials (The purpose of this separation is to avoid this file from getting committed in git and will remain local to the machine in which the code is running)

**/module/**

This is where the folder structure and the code for each module that needs to be used with the openemr are placed.

**ModuleName**/ *(Details of the folder content are explained separately)*

**/public/**

This is the default landing path. This will contain all the common files. Since different modules are being developed by different people the ideal structure of each of the sub folders in this folder should be a lib folder to contain all the re-usable lib files (css,js,image) and folders with name of the modules that will contain the module specific files

* .htaccess

All the rewrite rules are defined here

**/public/css/**

/lib

Common css files

/ModuleName

css files specific to module1

**/public/js/**

/lib

Common js files

/ModuleName

js files specific to module1

**/public/images/**

/lib

Common images

/ModuleName

Images specific to module1

**/tests/**

**/vendor/**

The vendor sub-directory should contain any third-party modules or libraries on which your application depends. This might include Zend Framework, custom libraries from your organization, or other third-party libraries from other projects. Libraries and modules placed in the vendor sub-directory should not be modified from their original, distributed state.

ZEND Module **(/interface/modules/zend\_modules)**

Each of the modules that are created under the module folder should follow the below structure

module\_root<named-after-module-namespace>/

Module.php

autoload\_classmap.php

autoload\_function.php

autoload\_register.php

config/

module.config.php

public/

images/

css/

js/

src/

<module\_namespace>/

<code files>

test/

phpunit.xml

bootstrap.php

<module\_namespace>/

<test code files>

view/

<dir-named-after-module-namespace>/

<dir-named-after-a-controller>/

<.phtml files>

**/ModuleName/Module.php**

A class under the module namespace which is eventually consumed by module manager to perform a number of tasks such as  setting up module-specific event listeners, auto loading classes, service configuration, view helper configuration, etc.

**/ModuleName/autoload\_classmap.php**

Return an array class map of class name/filename pairs (with the filenames resolved via the \_\_DIR\_\_ magic constant).

**/ModuleName/autoload\_function.php**

Return a PHP callback that can be passed to spl\_autoload\_register(). Typically, this callback should utilize the map returned by autoload\_classmap.php.

**/ModuleName/autoload\_register.php**

Register a PHP callback (is typically returned by autoload\_function.php with spl\_autoload\_register().

The point of these three files is to provide reasonable default mechanisms for auto loading the classes contained in the module.

**/ModuleName/config**

The config directory should contain any module-specific configuration. These files may be in any format Zend\Config supports. We recommend naming the main configuration “module.format”, and for PHP-based configuration, “**module.config.php**”. Typically, you will create configuration for the router as well as for the dependency injector. Default landing page in each of the module should be index (as expected by module installer). This will have configuration for controller, model and view for this module.

Hooks of Module can also be configured here.

**/ModuleName/public**

The public directory can be used for assets that you may want to expose in your application’s document root. These might include images, CSS files, JavaScript files, etc specific to a module. How these are exposed is left to the developer.

**/ModuleName/src/**

The src directory contains your module’s source code.

**ModuleName/**

**/Controller**

Contains the controller files

**/Form**

The forms classes should be maintained here

**/Model**

This will contain the class files representing the tables and a ModelNameTable file class where the model’s db operations that can be done.The table file should be the only place where the queries are defined and we are using the OpenEMR libraries to execute the queries. Each of the tables used can be represented as a class and a data retrieved from a function in the ModelTableClass can be converted to this object for use in controllers and views.

**/ModuleName/test/**

The test directory should contain your unit tests. Typically, these are written using [PHPUnit](http://phpunit.de/), and contain artifacts related to its configuration (e.g., phpunit.xml, bootstrap.php).

**/ModuleName/view/**

**ModuleName /**

**ControllerClassName/**

Contain template .phtml files for all the actions defined in the controller

**layout/**

This will contain the layout files for the module.

# Dependency Configuration

**/ModuleName/src/ ModuleName/Model/Configuration.php**

If any other module or modules is required to be installed and enabled for installing and enabling a particular module, we can specify such modules by its directory name as array elements in the function.

public function getDependedModulesConfig() {

//SPECIFY LIST OF DEPENDED MODULES OF A MODULE

$dependedModules = array('Encounter',

'Lab',

);

return $dependedModules;

}

Once the depended modules are configured (in the configuration file), it will check the depended modules whenever the user enables or disables the particular module.

# Hooks Configuration

Hooks help the user define links of the module into other parts of the system. We can create or configure hooks in a module as array values in a function “getHookConfig()” in a model file;

**/ModuleName/src/ ModuleName/Model/Configuration.php**

public function getHookConfig() {

//NAME KEY SPECIFIES THE NAME OF THE HOOK FOR UNIQUELY IDENTIFYING IN A MODULE.

//TITLE KEY SPECIFIES THE TITLE OF THE HOOK TO BE DISPLAYED IN THE CALLING PAGES.

//PATH KEY SPECIFIES THE PATH OF THE RESOURCE, SHOULD SPECIFY THE CONTROLLER

AND IT’S ACTION IN THE PATH, (INCLUDING INDEX ACTION)

$hooks = array (‘0’ => array (

'name' => "procedure\_order",

'title' => "Procedure Order",

'path' => "public/lab/index",

),

'1' => array (

'name' => "procedure\_result",

'title' => "Lab Result",

'path' => "public/lab/result/index",

),

);

return $hooks;

}

**ACL Configuration**

Access Control Lists (ACL) is a list of access control entities, each entity specifies the access rights allowed or denied. We can create or configure acl in a module as array values in a function “getAclConfig()” in a model file;

**/ModuleName/src/ ModuleName/Model/Configuration.php**

public function getHookConfig() {

//'section\_id : Specifies the id of the acl section for uniquely identifying in a module.

// name : Specifies the name of the acl section.

// parent\_section :Specifies parent section in tree view.

public function getAclConfig()

{

$acl = array(

array(

'section\_id' => 'lab',

'section\_name' => 'LAB',

'parent\_section' => '',

),

array(

'section\_id' => 'button\_1',

'section\_name' => 'Button 1',

'parent\_section' => 'lab',

),

);

return $acl;

}

**Settings Configuration**

Settings configuration allows the user to set the settings related to that particular module. The items shown in this area is defined by the creator of the module. It can be defined by creating the required items as a form in **Model/Configuration** by extending the zend form. The form is defined in the constructor of the class as shown below.

public function \_\_construct()

{

parent::\_\_construct('configuration');

$this->setAttribute('method', 'post');

$this->add(array(

'name' => 'hie\_path',

'attributes' => array(

'type' => 'text',

'id' => 'hie\_path'

),

'options' => array(

'label' => '

HIE Path',

),

));

$this->add(array(

'name' => 'test',

'type' => 'Zend\Form\Element\Select',

'attributes' => array(

'class' => '',

'data-options' => 'required:true',

'editable' => 'false',

'required' => 'required',

'id' => 'test'

),

'options' => array(

'label' => 'Test Options',

'value\_options' =>array(

'' => '--Select--',

'1' => 'Testing 1',

'2' => 'Testing 2',

),

),

));

$this->add(array(

'name' => 'test\_path',

'attributes' => array(

'type' => 'text',

'id' => 'test\_path'

),

'options' => array(

'label' => '

Test Path',

),

));

}

CUSTOM Modules **(/interface/modules/custom\_modules)**

This is rather a free form structure, except for some of the mandatory files.

**/ModuleName/**

index.php - The default landing page**(required)**

info.txt - This file should contain the module name**(required)**

table.sql - Sql’s to be run to get this module working**(required)**

moduleSettings.php –

Containing additional settings like ACL, HOOKS etc for the module. Settings are defined in an array named $ MODULESETTINGS and it is explained below.

$MODULESETTINGS = array(

'ACL' =>array(),

'hooks' => array()

);

‘ACL’ is used for the ACL settings for this module

‘ACL’=>array(

‘obj\_name’=>array(

‘menu\_name’=>’Display Name for the menu’

)

)

‘HOOKS’ is used for the HOOKS settings for this module.

HOOKS are used to use a particular page in Reports or

Encounter

‘HOOKS’=>array(

‘obj\_name’=>array(

‘menu\_name’=>’Display Name for the menu’,

‘path’=>’Path to the particular page from zend\_module or custom\_module’

)

)

**/ModuleName/public/**

**css/**

lib/ - css libraries used

style1.css - other css files

**js/**

lib/ - js libraries

script1.js - other js files

**images/**

lib/ - img libraries

img1.jpg - other img files

**config/** - add configuration files, if any

**lib/** - this can contain third party php libraries or php scripts that are common.

otherphp files