

How to

The Zend Framework may or may not be an easy framework for PHP development, but a GemsTracker project can be adapted without deep knowledge of the Framework. Of course when a project needs extensive adaptations (e.g. a separate survey engine or intensive integration with other applications) knowledge of the Zend Framework becomes necessary, but there is no need to dive in at the deep end of the framework.

Here are some examples of common extensions.

How do I get started

Follow the steps in the [Quickstart guide](#).

How do I debug the code

When you start nosing around in the code or start making changes to the code you often want to display some feedback on the page about what is going on or the content of variable.

A general method to start using code is to uncomment the `_initZFDebug()` function in `GemsEscort.php`.

How do I create my own controller

Check Zend FW Quickstart for an intro on controllers and their names and to get an introduction to the Zend naming conventions and project setup.

Most GemsTracker controllers use a Model (i.e. display and browse one set of data) and inherit from either `Gems_Controller_ModelSnippetActionAbstract` (the newer solution) or `Gems_Controller_BrowseEditAction` (old solution). Check `Gems_Default_StaffAction` for an example of the first and `Gems_Default_GroupAction` for the second type of standard controller.

In other cases, e.g. when you just want to output some (mostly) fixed HTML, you can use `Gems_Controller_Action` as a template. `Gems_Default_ContactAction` is a good example.

Of course you can also use your existing Zend controller. All Zend application variables will be set

Do not mirror the directory location from the Gemstracker library. The `classes/Gems/Default` location is used to ease the adaptation of existing controllers at the project level. Use instead the Zend Framework method of creating `xxxController` with same filename and object name in `application/controllers`.

How do I add my own controller to the menu?

You successfully created your `HelloController`?, but the you called the `World` action and Gems tells you

you are not allowed to access the page. This is because all access to pages is controlled through the Menu object. GemsTracker has a default menu, but you can change it to suite your needs.

Go to the `/application/classes/[project name]/Menu.php` file and create/edit the `loadProjectMenu()` function.

```
public function loadProjectMenu()
{
    // Hello world page
    $this->addPage('Hello', null, 'hello', 'world');
}
```

And voila: 'Hello' appears as a menu choice and you can select the controller.

The definitive place to check the workings of the `addPage()` function is of course the API documentation (for `Gems/Menu/MenuAbstract->addPage()`) or to load the source in your program editor. But here is how the `addPage` function works.

parameters label The label for display in the menu, leave null when used, but not displayed in the menu. privilege When empty the action is always accessible, specify 'pr.islogin'/'pr.nologin' when a user must/must not be logged in. Specify your own string when you want to set the privilege yourself for specific application roles. controller The name of the new controller. action The name of the action or 'index' by default. other An optional array for advanced usage. You should really have a look at the API documentation. return: a `Gems_Menu_SubMenuItem` object where you can specify sub items So putting it together you can add something more complicated.

```
public function loadProjectMenu()
{
    // Hello world page
    // - always accessible
    // - move to top of menu using 'order'
    $page = $this->addPage('Hello', null, 'hello', 'index', array(
        'order' => 0 // Put this page at the top of the menu
    ));

    // Add sub-page for logged in user
    $page->addPage('You', 'pr.islogin', 'hello', 'you');

    // Add sub-page for when not logged in
    $page->addPage('World', 'pr.nologin', 'hello', 'World');

    // Add sub-page for roles that you gave the my.secret privilege
    $page->addPage('Secret', 'my.secret', 'hello', 'secret');

    // Add sub-page for everybody
    $page->addPage('Everybody', null, 'hello', 'everybody');

    // Add sub-page that is not displayed, but that you can access when
    you know the url
    $page->addPage(null, null, 'hello', 'hidden');
}
```

How do I change the displayed columns in the respondent/patient overview?

As every project has it's own data to work on, one of the most common actions is to change the display of the columns shown in the Respondents/Patients screen.

First to explain why we sometimes talk about respondents and sometimes about patients. Of course a respondent is someone who either answers a survey or about whom a survey is answered by someone else. Either way the survey belongs to that respondent. A patient is of course someone who receive health care. As GemsTracker is build for health care institutions in most existing projects respondents are patients, but there is no reason why they could not be truckers, butchers, mail man or just any other group of random respondents. That is why internally GemsTracker reverts to respondents (we even hunt down the use of the word patient in the core library) but use a default English translation that does not but translate 'respondent' into 'patient'.

To change the displayed columns copy library/Gems/controllers/RespondentController.php' to application/controllers/RespondentController.php'. GemsTracker automatically sees the new RespondentController.php file and starts using that controller. RespondentController.php is an empty stub that inherits all functionality from Gems_Default_RespondentAction. This allows you to overrule or extend the default functionality without have to copy the code, with all the maintenance issues this entails.

Now your first thought might be that to change the workings of the 'index' action you must overrule the indexAction() function, but the search as you type function returns a result from autofilterAction() and column content is added in the protected _createTable() function that is used by both functions. However, the actual action of specifying what columns have to be displayed is done in the addBrowseTableColumns() function overloaded in RespondentAction. To change the columns you must overload this function again in your copy of RespondentController.

Now you might have noticed that the Respondent index screen displays quite a complicated table. The data is displayed in rows, but most cells have two lines in them, e-mail addresses are links when they exists, there are some extra texts that appear only when needed. There is a paginator, you can change the number of rows, the headers are sort links. There are buttons on the sides and if you click on a row, it acts as if the first button was clicked. You noticed all that? Thank you! Did you also notice that you can use the Page Up and Page Down keys to browse, Ctrl Up and Ctrl Down to see more or less rows, and use the normal arrow keys to select a row and then press enter to open it? Thought not. Well that all works too.

It must take hundreds of lines of code to write it, no? Well yes, it does take hundreds of lines, but if we look in RespondentAction we see the code actually written in the function is quite limited. All the magic work is done by the TableBridge and AbstractModel objects.

```
protected function addBrowseTableColumns(MUtil_Model_TableBridge
$bridge, MUtil_Model_ModelAbstract $model)
{
    $model->setIfExists('gr2o_opened', 'tableDisplay', 'small');
    $model->setIfExists('grs_email', 'itemDisplay',
'MUtil_Html_AElement::ifmail');

    if ($menuItem = $this->findAllowedMenuItem('show')) {
$bridge->addItemLink($menuItem->toActionLinkLower($this->getRequest(),
$bridge));
    }
```


From:

<https://gemstracker.org/wiki/> - **GemsTracker**

Permanent link:

<https://gemstracker.org/wiki/doku.php?id=devpage:howto&rev=1345726813>

Last update: **2020/03/12 12:08**

