Forms in Zend Framework. Getting rid from routines.

Introduction

Forms and form handling are important parts of the web development. Form handling includes the following steps:
- Form displaying
- Form data filtering
- Form data validation

Zend Framework provides very powerful component for forms handling. In a Zend Framework based application each form has it's own class where form attributes are set, rules for filtering and validation are described. Please see the Listing 1 for the example

Listing 1. Zend_Form class example

```
class forms_ContactForm extends Zend_Form
{
   public function __construct($options = null)
   {
      parent::__construct($options);
      $this->setName('contact_us');

      $firstName = new Zend_Form_Element_Text('firstName');
      $firstName->setLabel('First name')
            ->setRequired(true)
            ->addValidator('NotEmpty');

      $email = new Zend_Form_Element_Text('email');
      $email->setLabel('Email address')
          ->addFilter('StringToLower')
          ->setRequired(true)
          ->addValidator('NotEmpty', true)
          ->addValidator('EmailAddress');
      $submit = new Zend_Form_Element_Submit('submit');
      $submit->setLabel('Contact us');

      $this->addElements(array($title, $firstName,
        $lastName, $email, $submit));
  }
}
```

So as you can see Zend_Form provides you abilities to manipulate form fields, to filter and

validate form data.

Let's say that you are working on a large project with lots of forms. You have no choice but to write classes describing all of these forms. You can be sure these classes will have much in common. Each class will have a code to create a form element, assign filters and validations to the element. When it is needed to create a bunch of similar class I would call it a routine.
It's not effective to spend a lot of time on routine tasks so it's better to think of how to automate them.

Defining an approach

One of the possible solutions is to create a configuration file for each form. But you would still have to describe each form element manually, also these configuration files should be parsed to handle forms and this will require additional server resources. So my suggestion is to create a tool for form classes generation. It's possible to create a web interface for this tool and use it via browser but there is an easier and more elegant from my point of view way to solve this problem. Let's make a console application and it will be easy to do it because Zend Framework already has everything we need for it. I would like to mention Zend_CodeGenerator and Zend_Tool components.

Generate code with Zend_CodeGenerator

Zend_CodeGenerator component allows to generate php code. It is rather simple to use the component so I would not explain how to do it. Just visit http://framework.zend.com/manual/en/zend.codegenerator.html for details. Lets use this component to generate code for our forms. Listing 2 demonstrates how class could be generated

Listing 2

```
        $codeGenFile = new Zend_CodeGenerator_Php_File(array(
                'classes' => array(
                        new Zend_CodeGenerator_Php_Class(array(
                                'name' => $className,
                                'extendedClass' => 'Zend_Form',
                                'methods' => array(
                                new Zend_CodeGenerator_Php_Method(array(
                                        'name' => '__construct',
                                        'body' => $body,
                                        'parameters'=>array(array('defaultValue'=>array(),
'name'=>'options')),
                                ))
```

```
                                    )
                            ))
                    )
            ));
```

The code above will generate class with name $className. This class will be extended from Zend_Form and will contain __construct() method. The only problem left to solve is to generate **$body** the content of the __construct().

Console application with Zend Framework

Zend_Tool component allows to develop console applications. The development process is rather easy and includes the following steps:
1. Create a provider class (This class will encapsulate your application logic)
2. Create a manifest class for your provider (It is info class which will display the information about your provider). You could skip this step but it's better not to do it as it is not a good rule.
3. Tell Zend Framework where you provider and manifest are located

Listing 3 shows an example how provider could be created

Listing 3. Provider example

```
class My_Tool_Project_Provider_MyProvider implements
Zend_Tool_Framework_Provider_Interface{
   public function getName()
   {
      return 'my_provider';
   }

   public function say()
   {
      echo "Hello From Demo Provider!";
   }
}
```

And Listing 4 shows how to create manifest

Listing 4. Manifest example

```
class My_Tool_Project_Provider_Manifest implements
Zend_Tool_Framework_Manifest_ProviderManifestable
{
```

```
    public function getProviders()
    {
        return array(
                new My_Tool_Project_Provider_MyProvider
        );
    }
}
```

So we have provider and manifest done. Let's tell Zend Framework where these classes could be found. We just need to add these classes into Zend_Tool configuration file. By default this file should be located in your home folder and it should have .zf.ini filename.
If this file is missing you could create it. Just go to bin folder (it is bundled with Zend Framework) and run:

```
> cd /your/project/path/bin
> ./zf.sh create config
```

As the result .zf.ini file will be created. It's possible to store this configuration file anywhere you want just set **ZF_HOME** environmental variable into desired value. In this case your configuration file path will be **ZF_HOME/.zf.ini**

Once you've created this file just open it with text editor and add the following data:

```
basicloader.classes.0 = "My_Tool_Project_Provider_MyProvider"
basicloader.classes.1 = "My_Tool_Project_Provider_Manifest"
```

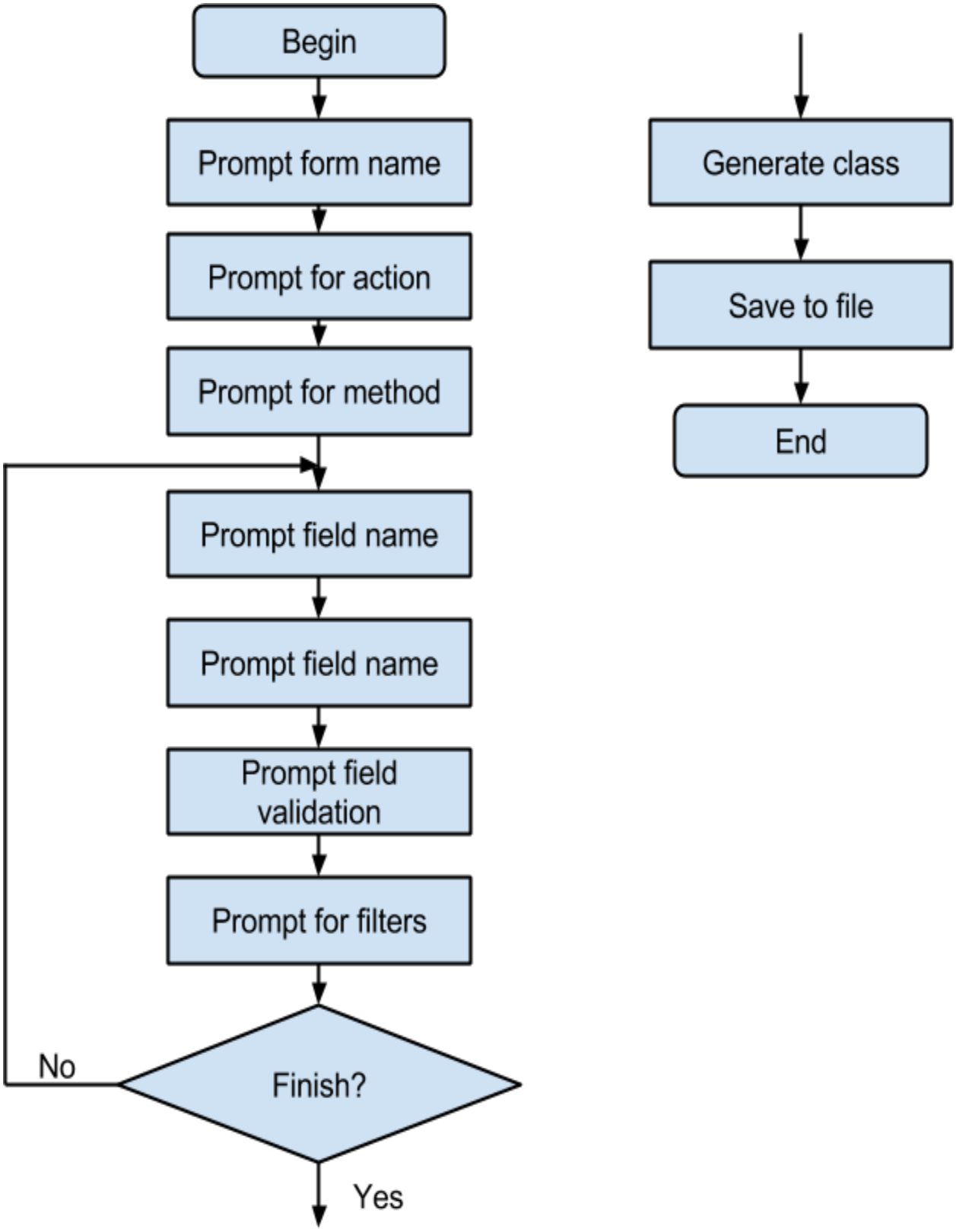
Let's test everything

```
> cd /your/project/path/bin
> ./zf.sh say my_provider
```

If everything is correct you'll see "Hello From Demo Provider!" output.

Developing the tool

Once we know how to create a console application and how to generate a code let's develop our tool. The following figure describes the algorithm of the application. As you could see it is really simple. The source code is provided in the attachment so you could review and try it.


Begin
Prompt form name
Prompt for action
Prompt for method
Prompt field name
Prompt field name
Prompt field validation
Prompt for filters
Finish?
No
Yes
Generate class
Save to file
End

So we have developed the provider which is called My_Tool_Project_Provider_Forms. This provider has **generate()** method which is used to generate a form class. Once we add this class into .zf.ini we could try it.

cd /your/project/path**/bin**
./zf.sh generate forms
“Please provide a value for $formName” is displayed
zf > name_of_the_form
“Please provide a value for $method” is displayed
zf > POST
“Please provide a value for $action” is displayed
zf > /form_action.php
Are you done?
0. Yes
1. No

zf > 0 || 1 # press 0 to finish and 1 to continue
“Type field name“ is displayed in case you’ve been chosen 1
zf > field_name
Type field label
zf> Field Label
“Choose field type” is displayed with the list of available field types
zf > #choose field type by typing its number

Then system will ask if the field is required after that you’ll be offered to set validation rules and then you could finish or add another field. Once you choose to finish the form class will be generated in the **/your/project/path/application/forms** folder.

Conclusion

Application we've just created allows to generate classes on the basis of our input. This helps us to avoid routine coding but we still have files with classes which could be edited in case we would need some tweaks or changes. Also we've excluded the following steps in form programming:

- Create and name class with form manually
- Write class skeleton. Also there is no need to think how to name class according to the naming convention because name is generated automatically
- Describe form fields using Zend_Form set of functions

From my own experience an average time of form creation has decreased from 25 minutes to 10. This means that you are able to save time to develop other features of your project or devote this time to testing or bug fixing. Anyway this could help you to deliver more quality code in shorter time and this is always great.

References

http://framework.zend.com/manual/en/zend.form.html
http://framework.zend.com/manual/en/zend.codegenerator.html
http://framework.zend.com/manual/en/zend.tool.html