

# Working with Zend\Form

Rob Allen

ZendCon, October 2012

# The problem with forms

They break the concept of separation of concerns!

# ZF2's form solution

Multiple parts working together

# Zend\Form

A bridge between the model entity and the view

- elements & fieldsets
- input filter
- data transfer methods

# Elements

- Button
- Captcha
- Checkbox
- Collection
- Color
- Csrft
- Date
- DateTime
- DateTimeLocal
- Email
- File
- Hidden
- Image
- Month
- MultiCheckbox
- Number
- Password
- Radio
- Range
- Select
- Submit
- Text
- Textarea
- Time
- Url
- Week

# Zend\InputFilter

Provides:

- Filtering of input data
- Validation of input data

# View helpers

- Render a form using a series of view helpers:
  - `form( )` for open and close tags
  - `formXxx( )` for each element: ie: `formHidden( )`
  - `formLabel( )` for each element's label
  - `formElementErrors( )` for validation messages
  - Convenience helpers `formInput( )`, `formRow( )`
- No decorators: you have control!

# Data transfer

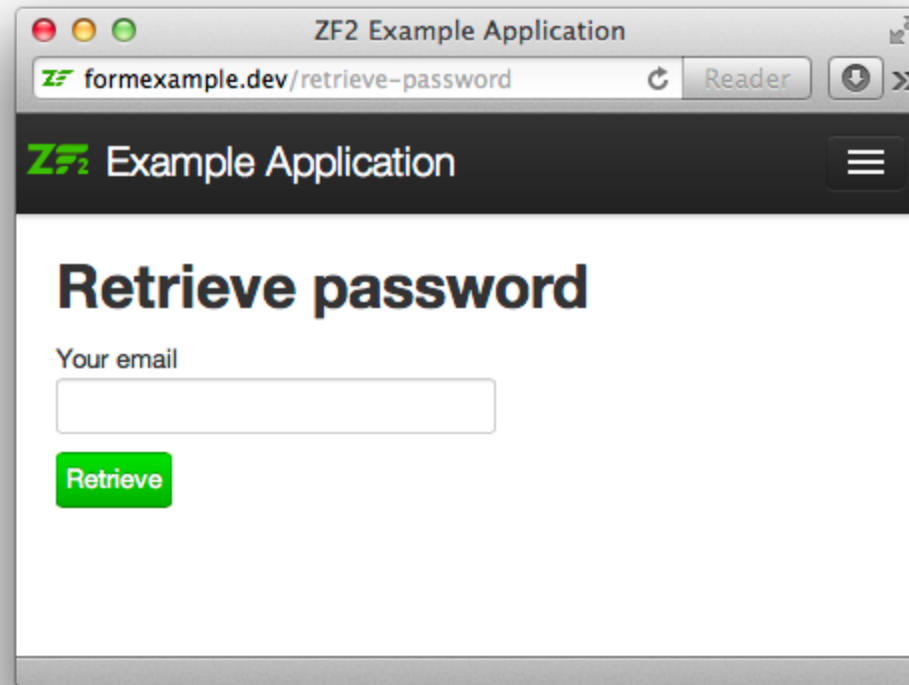
- bind an object to the form, then it will:
  - Map validated values into object after post
  - Extract values from object to populate form
- Multiple strategies:
  - ArraySerializable
  - ObjectProperty
  - ClassMethods



# Zend\Form in use!

(Warning: lots of code coming up!)

# Consider this



# The HTML

```
<h1>Retrieve password</h1>
<form action="/" method="POST">
<label for="email">Your email</label>
<input type="text" name="email" id="email">
<input type="submit" name="submit"
      class="btn-success" value="Retrieve">
</form>
```

# Define a form

```
namespace Application\Form;
use Zend\Form\Form;
class RetrievePwd extends Form
{
    public function __construct($name = null)
    {
        parent::__construct('retrievepwd');

        /* Elements defined here */
    }
}
```

# Email elements

```
// within RetrievePwd::__construct()
$this->add(array(
    'name' => 'email',
    'type' => 'Zend\Form\Element\Email',
    'options' => array(
        'label' => 'Your email',
    ),
    'attributes' => array(
        'id' => 'email',
    ),
));
```

# Submit Element

```
$this->add(array(  
    'name' => 'submit',  
    'type' => 'Zend\Form\Element\Submit',  
    'attributes' => array(  
        'value' => 'Retrieve',  
        'class' => 'btn-success',  
    ),  
));
```

# Controller code

```
namespace Application\Controller;
use Zend\Mvc\Controller\AbstractActionController;
use Application\Form\RetrievePwd as RPForm;
class IndexController
    extends AbstractActionController
{
    public function newPwdAction()
    {
        $form = new RetrievePwdForm();

        return array('form' => $form);
    }
}
```

# View script

```
// in view/index/new-pwd.phtml
$form = $this->form;
$form->setAttribute('action',
    $this->url('new-pwd'));
$form->prepare();

echo $this->form()->openTag($form) . "\n";
echo $this->formLabel($form->get('email')) . "\n";
echo $this->formInput($form->get('email')) . "\n";
echo $this->FormElementErrors($form->get('email'));
echo $this->formSubmit($form->get('submit')) . "\n";
echo $this->form()->closeTag($form) . "\n";
```



# Using formRow

The `formRow` view helper simplifies rendering of elements:

```
echo $this->formLabel($form->get('email'))."\n";  
echo $this->formInput($form->get('email'))."\n";  
echo $this->FormElementErrors($form->get('email'));
```

becomes:

```
echo $this->formRow($form->get('email'))."\n";
```

# formRow: the view script

```
// in view/index/new-pwd.phtml
$form = $this->form;
$form->setAttribute('action', $this->url('new-pwd'));
$form->prepare();

echo $this->form()->openTag($form) . "\n";
echo $this->formRow($form->get('email')) . "\n";
echo $this->formRow($form->get('submit')) . "\n";
echo $this->form()->closeTag($form) . "\n";
```

# or use formCollection

```
// in view/index/new-pwd.phtml
$form = $this->form;
$form->setAttribute('action', $this->url('new-pwd'));
$form->prepare();

echo $this->form()->openTag($form) . "\n";
echo $this->formCollection($form) . "\n";
echo $this->form()->closeTag($form) . "\n";
```

# Validation: InputFilter

Attach an `InputFilter` to the form:

```
namespace Application\Form;
use Zend\InputFilter\InputFilter;

class RetrievePwdFilter extends InputFilter
{
    public function __construct()
    {
        /* Input objects defined here */
    }
}
```

# Validation: Inputs

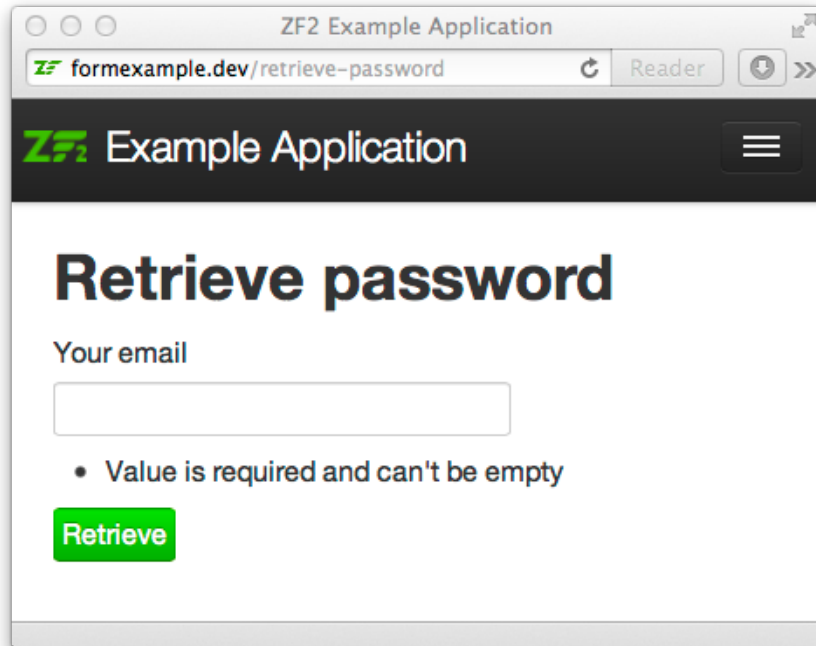
```
// Within RetrievePwdFilter::__construct():
$this->add(array(
    'name'      => 'email',
    'required' => true,
    'filters'   => array(
        array('name' => 'StringTrim'),
    ),
    'validators' => array(
        array('name' => 'EmailAddress'),
    ),
));
```

# Controller code

Attach input filter and process POST'ed request

```
public function newPwdAction() {
    $f = new RetrievePwdForm();
    $f->setInputFilter(new RetrievePwdFilter());
    if ($this->getRequest()->isPost()) {
        $f->setData($this->getRequest()->getPost());
        if ($f->isValid()) {
            // do something useful
        }
    }
    return array('form' => $f);
}
```

# Errors



ZF2 Example Application

formexample.dev/retrieve-password

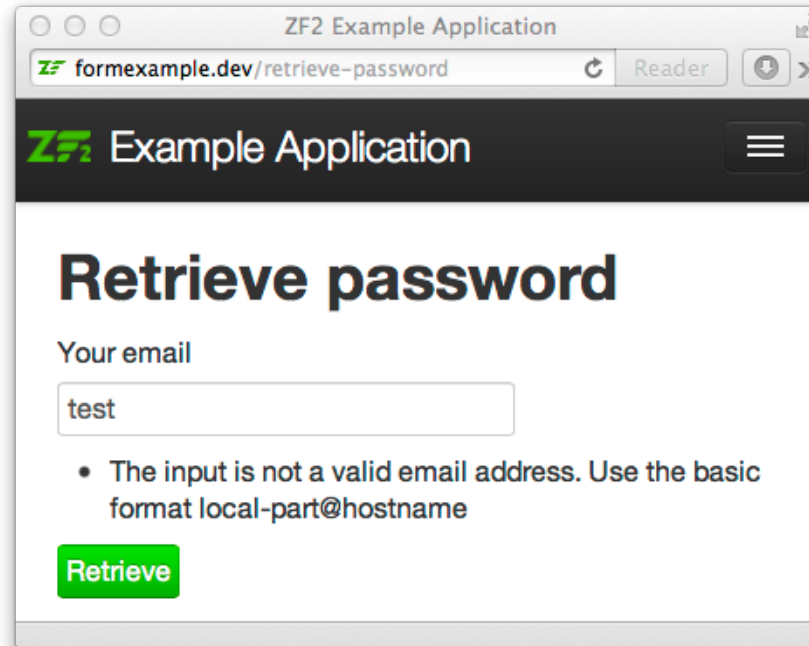
Example Application

## Retrieve password

Your email

- Value is required and can't be empty

Retrieve



ZF2 Example Application

formexample.dev/retrieve-password

Example Application

## Retrieve password

Your email

- The input is not a valid email address. Use the basic format local-part@hostname

Retrieve

# Change the error message

```
// 'validators' key in RetrievePwdFilter:
'validators' => array(
  array(
    'name'      => 'EmailAddress',
    'options'   => array(
      'messages' => array(
        'emailAddressInvalidFormat'
          => 'Your email address is invalid'),
      ),
    ),
  ),
),
```



# The "required" message

```
// 'validators' key in RetrievePwdFilter:
'validators' => array(
    array(
        'name' => 'NotEmpty',
        'break_chain_on_failure' => true,
        'options' => array(
            'messages' => array('isEmpty'
                => 'Email address is required'),
        )
    ),
    array(
        'name' => 'EmailAddress',
        'options' => array( /* ... */ ),
    ),
),
```

# Empty required field

Use `allow_empty` when a required field may be an empty string:

```
$this->add(array(
    'name'      => 'email',
    'required'  => true,
    'allow_empty' => true,
    'filters'   => array(
        array('name' => 'StringTrim'),
    ),
    'validators' => array( /* ... */ ),
```

# Data Transfer: binding

Binding allows the form to:

- set initial values the form fields from an object
- populate that object with the submitted data
- usually used for transferring all fields to a model entity

# Data Transfer: binding

```
public function newPwdAction()
{
    $form = new RetrievePwdForm();
    $form->setInputFilter(new RetrievePwdFilter());

    $sendPwdSvc = new SendPwdService();
    $form->bind($sendPwdSvc);

    // cont...
```

# Data Transfer: binding

```
// cont...
$request = $this->getRequest();
if ($request->isPost()) {
    $form->setData($request->getPost());
    if ($form->isValid()) {
        $sendPwdSvc->sendLostPasswordEmail();
        // redirect somewhere useful
    }
}
return array('form' => $form);
}
```

# ArraySerializable Hydrator

```
class SendPwdService
{
    protected $email;
    public function getArrayCopy()
    {
        return array('email' => '');
    }
    public function populate($data)
    {
        $this->email = $data['email'];
    }
    public function sendLostPwdEmail() { /* ... */ }
}
```

# ArraySerializable Hydrator

Or use `getArrayCopy()` and `exchangeArray()`. These are implemented by `ArrayObject`:

```
namespace Application\Service;

class SendPwdService extends \ArrayObject
{
    public function sendLostPasswordEmail()
    {
        // do work with $this['email']
    }
}
```

# Other hydrators

## ObjectProperty:

```
protected $email
```

## ClassMethods:

```
function getEmail()  
function setEmail($email)
```



**That's all you really need to know**

But there's more...

# Zend\Form\Fieldset

- *Logically* group elements and bind objects to them
- Ideal for groups of elements for each model entity
- Also handles One to One and One to Many relationships in your forms
- Not related to rendering in a `<fieldset>` (but you can do!)
- Arguably mis-named!

# E.g: reporting a bug

Consider this entity

```
class Bug
{
    public $title = '';
    public $description = '';

    // other properties: date, priority, etc.
}
```

# A 'Bug' Fieldset

```
class BugFieldset extends Fieldset
{
    public function __construct($name = null)
    {
        parent::__construct('bug');
        $hydrator = new ObjectPropertyHydrator();
        $this->setHydrator($hydrator);

        $this->add(array(
            'name' => 'title',
            'options' => array('label' => 'Title'),
        ));
        // cont...
```

# A 'Bug' Fieldset

```
// cont...
$this->add(array(
    'name' => 'description',
    'type' => 'Zend\Form\Element\Textarea',
    'options' => array(
        'label' => 'Description'),
));
// other elements: date, priority, etc.
}
}
```

# Adding an input filter

```
class BugFieldset extends Fieldset
  implements InputFilterProviderInterface
{
  // after __construct()
  public function getInputFilterSpecification()
  {
    return array(
      'title' => array('required' => true),
      'description' => array(
        'validators' => array('StripTags')
      ),
      // other inputs for date, priority, etc.
    );
  }
}
```

# We need a form too

```
class Bug extends Form
{
    public function __construct($name = null)
    {
        parent::__construct('bug');
        $this->setAttribute('method', 'post')

        $hydrator = new ObjectPropertyHydrator();
        $this->setHydrator($hydrator);

        // cont...
```



# Form

```
// cont...
$this->add(array(
    'type' => 'Application\Form\BugFieldset',
    'options' => array(
        'use_as_base_fieldset' => true)
));

$this->add(array(
    'type' => 'Zend\Form\Element\Csrf',
    'name' => 'csrf'
));
// and a submit button here
```

# View script

```
$form = $this->form;
$form->setAttribute('action', $this->url('bug'));
$form->prepare();
echo $this->form()->openTag($form);

// bug fieldset
echo "<fieldset class='bug'><legend>Bug</legend>";
$bug = $form->get('bug');
echo $this->formRow($bug->get('title'));
echo $this->formRow($bug->get('description'));
echo "</fieldset>";

// cont...
```

# View script

```
// cont...  
  
// CSRF element & submit button  
echo "<div class='submit'>";  
echo $this->formHidden($form->get('csrf'));  
echo $this->formSubmit($form->get('submit'));  
echo "</div>";  
  
echo $this->form()->closeTag($form);
```

# Binding the entity

```
public function bugAction()
{
    $form = new BugForm();
    $bug = new Bug();
    $form->bind($bug);

    if ($this->request->isPost()) {
        $form->setData($this->request->getPost());
        if ($form->isValid()) { /* save here */ }
    }

    return array('form' => $form);
}
```

**That was easy enough**

# Adding a reporter

Let's have a User class

```
class User {  
    public $name = '';  
    public $email = '';  
}
```

# Update the Bug class

```
class Bug {
    public $title = '';
    public $description = '';
    // other properties: date, priority, etc.

    public $reporter;

    function __construct()
    {
        $this->reporter = new User();
    }
}
```

# The reporter fieldset

```
class UserFieldset extends Fieldset
    implements InputFilterProviderInterface
{
    public function __construct($name='')
    {
        parent::__construct($name);
        $hydrator = new ObjectPropertyHydrator();
        $this->setHydrator($hydrator);
        $this->setObject(new User());

        // cont...
```



# The reporter fieldset

```
// cont...
$this->add(array(
    'name' => 'name',
    'options' => array('label' => "Name"),
));
$this->add(array(
    'name' => 'email',
    'type' => 'Zend\Form\Element\Email',
    'options' => array('label' => "Email"),
));
}
// don't forget getInputFilterSpecification()
```

# Add the reporter to the bug

```
class BugFieldset extends Fieldset
    implements InputFilterProviderInterface
{
    public function __construct($name = null)
    {
        // after 'bug' elements

        $this->add(array(
            'type' => 'Application\Form\UserFieldset',
            'name' => 'reporter',
        ));
    }
}
```

# Add to the view script

```
// after the bug fieldset

// reporter fieldset
$reporter = $bug->get('reporter');
echo "<fieldset><legend>Your details</legend>";
echo $this->formRow($reporter->get('name'));
echo $this->formRow($reporter->get('email'));
echo "</fieldset>" . PHP_EOL;
```

# That's it!

Note that we didn't need to worry about hydration and binding as it happens automatically

# Things I didn't cover

- Collections
- Annotations
- Validation groups

(You'll have to look these up yourself!)

# Thank you!

Please rate this talk: <http://joind.in/6876>

twitter: @akrabat

web: <http://akrabat.com>