MAKING THE MOST OF 2.2

MARK STORY

@MARK_STORY

PACKING IN THE GOOD

- 2.0.0 was release October 18, 2011
- Since then 2.1, 2.2, and 2.3 have been released or started.
- Over 20 releases since 2.0.0.
- Highest release velocity ever for CakePHP.

FOCUS ON PROBLEMS

We've tried to keep releases focused on solving real world problems developers have everyday.

Make upgrading as easy as possible.

VIEW BLOCKS

PROBLEM

- Multiple similar views would often contain repeated structural HTML.
- Apps suffered from element-itis to keep HTML DRY.
- Piles of elements make code hard to follow and understand.

- Inspired by blocks in Jinja/Twig.
- Allows you to create slots/blocks in layouts or parent templates.
- Helps keep views/layouts more DRY by letting you create extensible HTML content.
- Replaces annoying magic variables like `$scripts_for_layout` and `$content_for_layout`.

EXAMPLE

Parent wrapper view with a child view

PARENT VIEW

```
<h1 class="content-title"><?= $this->fetch('title'); ?></h1>
<div class="content">
<?= $this->fetch('content'); ?>
</div>
<div class="sidebar">
<?= $this->fetch('sidebar'); ?>
</div>
<?php if ($this->fetch('pagination')): ?>
<div class="pagination">
<?= $this->fetch('pagination'); ?>
</div>
<?php endif; ?>
```

CHILD VIEW

```
<?php $this->extend('../common/sidebar.ctp'); ?>
<?php $this->assign('title', 'Product list'); ?>
<?php $this->start('content'); ?>
<p>This is the content</p>
<?php $this->end(); ?>

<?php $this->start('sidebar'); ?>
<p>This is a sidebar</p>
<?php $this->end(); ?>
```

JSON & XML VIEWS

PROBLEM

- Previously creating JSON and XML views that just serialized data was a pain.
- Tons of repetitive views and layout files required.

- Two view classes that allow you to easily serialize data for simple uses.
- Special `_serialize` view variable defines which view variables should be serialized.
- You can also use normal view files if you need to massage data first.
- Integrates well with existing features like exception handling and RequestHandlerComponent.

EXAMPLE

```php
<?php
// In a controller.
function index() {
        $this->set('tasks', $this->paginate());
        $this->set('_serialize', array('tasks'));
}
```

HASH CLASS

PROBLEM

- `Set` is full of inconsistencies. Both in the API and path selector syntax.
- Set::extract() while powerful, is slow and insane inside.
- xpath-ish syntax full of un-fixable bugs and not supported in most methods.

- Hash implements >90% of Set's API.
- All methods have a consistent signature.
- The same dot notation features are supported everywhere.
- Up to 1.6x faster on extract().
- Most methods are faster as well.

PERFORMANCE COMPARISONS

Do a similar operation 1000 times, on the same data.

`Hash::extract($d, '{n}.Article.id')`	0.215131998
`Set::extract('/Article/id', $d)`	0.23719382

1.1x improvement

`Hash::extract($d, '{n}.Comment.{n}.id')`	0.173635005
`Set::extract('/Comment/id', $d)`	0.201920986

1.1x improvement

`Hash::maxDimensions($d)`	0.075587987
`Set::countDim($d, true)`	0.575523138

7.6x improvement

API CONSISTENCY

- Every method takes an array of data as the first argument.
- Second argument is always one or more paths.
- A consistent API is easier to document, understand and remember.

CAKETIME & CAKENUMBER

PROBLEM

- `TimeHelper`, `NumberHelper` and `TextHelper` have some very useful methods.
- But they are all trapped in a helper.

- Expose non HTML related features as a utility library.
- Greatly improve timezone handling.
- New features for testing & working with datetimes.

EXAMPLE

```php
<?php
App::uses('CakeTime', 'Utility');
App::uses('CakeNumber', 'Utility');

$utc = CakeTime::toServer($datetime, 'America/Toronto');
$today = CakeTime::isToday($timestamp);

$result = CakeNumber::toReadableSize($oneTerabyte);
```

LOGGING++

PROBLEM

- Granular logging based on application section was impossible.
- Log messages were simply broadcast to every connected logger.

- Borrowed idea of logging 'scopes' from python's `logging` module.
- Log messages can be tagged with scopes.
- Only loggers interested in those scopes will get scoped messages.

ATTACH A LOGGER WITH A SCOPE

```php
<?php
CakeLog::config('payments', array(
        'engine' => 'Email'
        'scopes' => array('payments', 'billing'),
        'to' => 'mark@example.com'
));
```

LOG MESSAGES WITH A SCOPE

```php
<?php
try {
        $gateway->process($payment);
} catch (PaymentException $e) {
        CakeLog::error(
                'Transaction failed ' . $e->getMessage(),
                array('payments', 'billing')
        );
}
```

NOW TO JOSE