

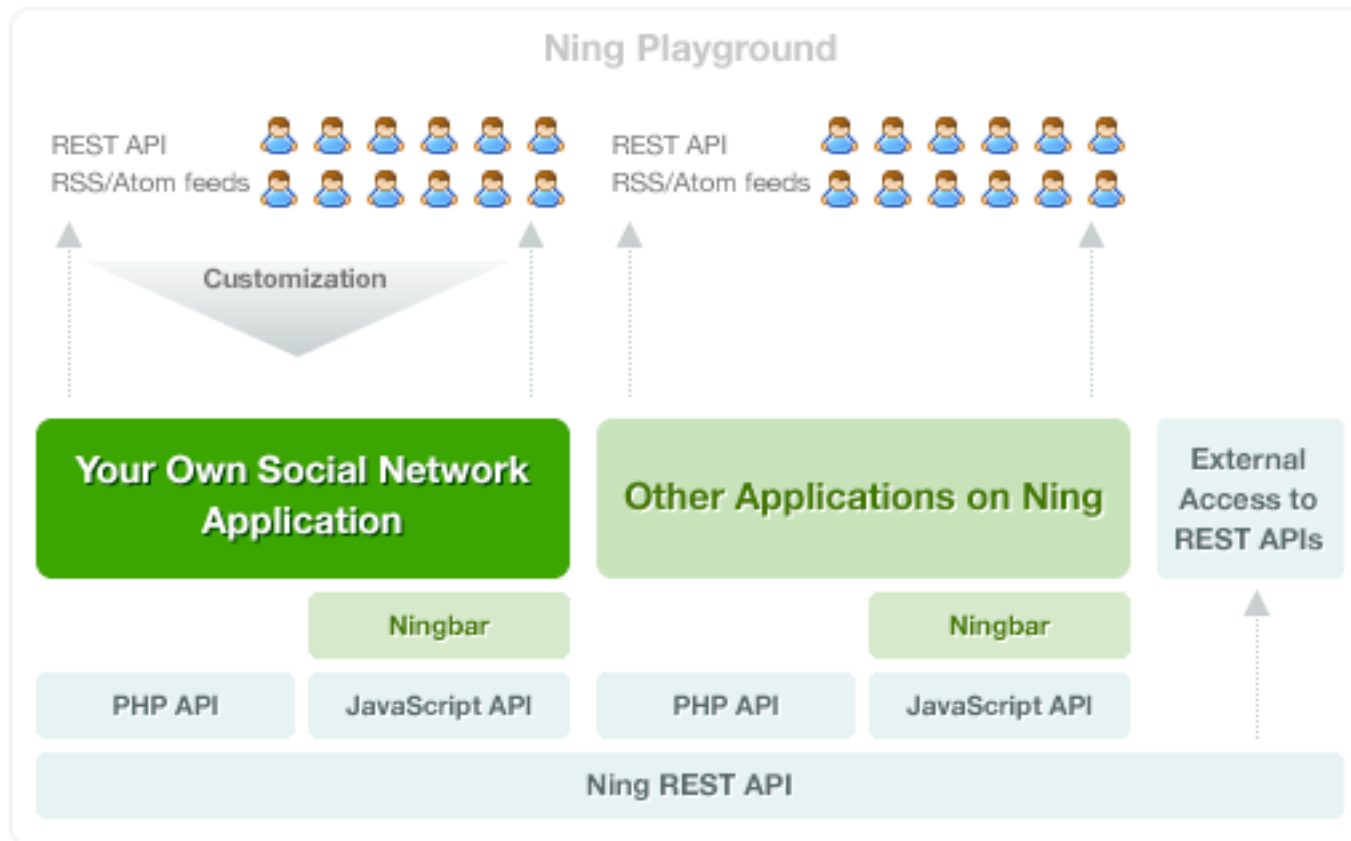
API Design in PHP

David Sklar

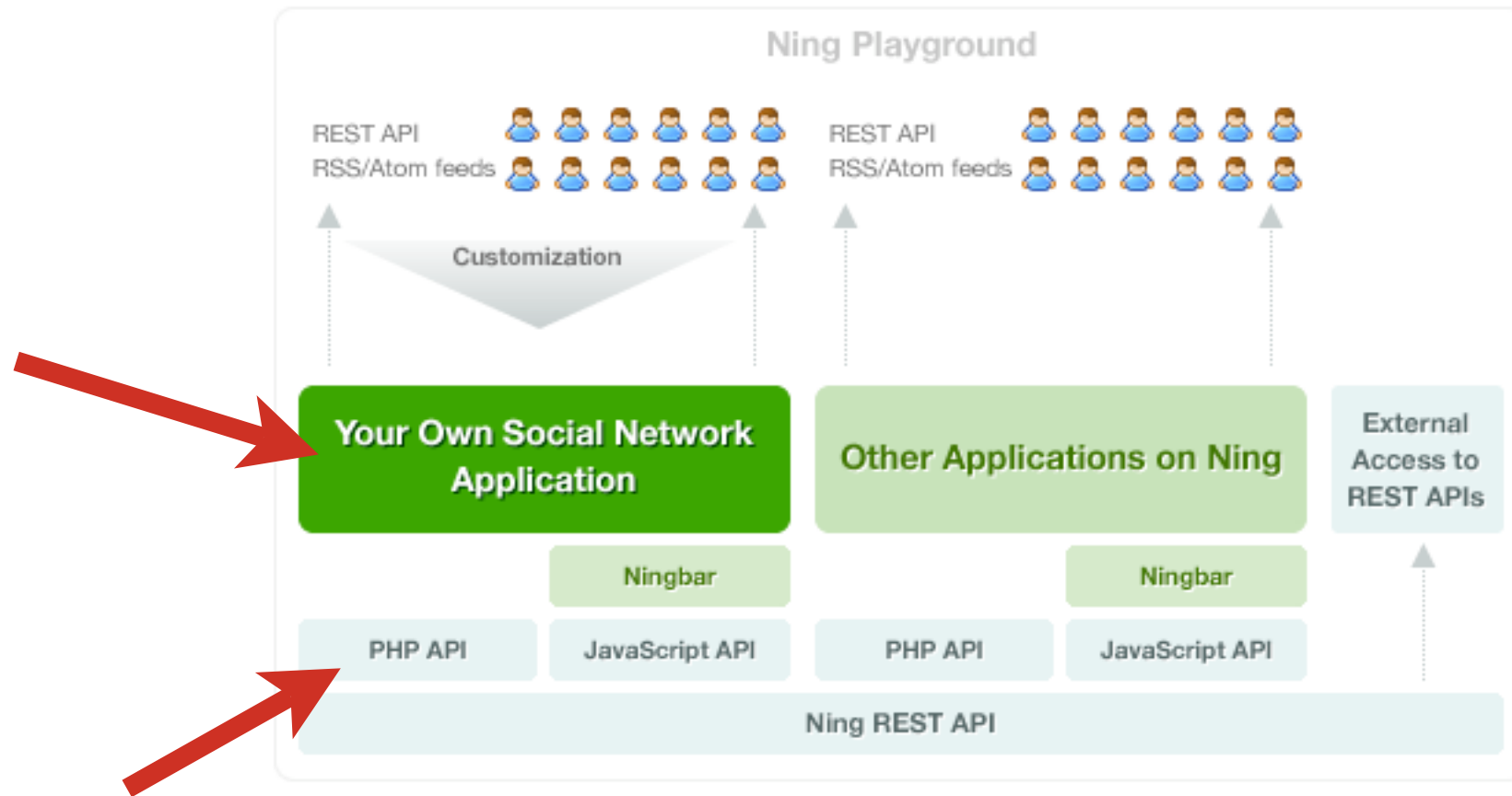
Software Architect, Ning Inc.

david@ninginc.com

DC PHP Conference 2007



Ning Platform



Ning Platform

Ning

- PHP API provides interface to our platform REST APIs
- Live since August 2005 (with 5.0.4)
- Recent upgrade to 5.2.3
- In use in all 118,000+ networks on the platform

Ning

- Migration from XMLRPC to REST in 2005/6
- APIs used for content storage, user profile management, tagging, search, video transcoding, messaging, ...
- PHP (using APIs) runs in a hosted environment

API: XN_Content

```
<?php
$dinner = XN_Content::create('Meal');
$dinner->title = 'Salt Baked Combo';
$dinner->my->protein = 'seafood';
$dinner->my->ingredients =
    array('shrimp', 'scallops', 'squid');
$dinner->save();
?>
```

PHP REST

POST /xn/atom/1.0/content

Content-Type: text/xml;charset=UTF-8

```
<entry xmlns="http://www.w3.org/2005/Atom"
        xmlns:xn="http://www.ning.com/atom/1.0"
        xmlns:my="http://afternoonsnack.ning.com/xn/atom/1.0">
  <xn:type>Meal</xn:type>
  <title type="text">Salt Baked Combo</title>
  <my:protein type='string'>seafood</my:protein>
  <my:ingredients type='string'>
    <xn:value>shrimp</xn:value><xn:value>scallops</xn:value>
    <xn:value>squid</xn:value>
  </my:ingredients>
</entry>
```

PHP REST

HTTP/1.1 200 OK

```
<?xml version='1.0' encoding='UTF-8'?>
<feed xmlns="http://www.w3.org/2005/Atom"
      xmlns:xn="http://www.ning.com/atom/1.0">
  <xn:size>1</xn:size>
  <updated>2007-08-28T22:11:47.420Z</updated>
  <entry xmlns:my="http://afternoonsnack.ning.com/xn/atom/1.0">
    <id>http://afternoonsnack.ning.com/502068:Meal:122</id>
    <xn:type>Meal</xn:type>
    <xn:id>502068:Meal:122</xn:id>
    <title type="text">Salt Baked Combo</title>
    <published>2007-08-28T22:11:47.414Z</published>
    <updated>2007-08-28T22:11:47.414Z</updated>
    <link rel="alternate"
          href="http://afternoonsnack.ning.com/xn/detail/502068:Meal:122" />
    <my:protein type="string">seafood</my:protein>
    ...
  </entry>
</feed>
```

Design Priorities

- Promote predictability, modularity, stability
- Choose human performance over computer performance
- Make efficiency easy, make inefficiency hard/impossible

At the start...

- Write code before you write the API
- Use cases, Use cases, Use cases
- Names matter (but don't discuss them forever)

Use the API before it exists

Sketch out what you want to do....

```
<?php
|
/* Load a profile by screen name, as before */
$profile = XN_Profile::load('screenName');
/* Load a profile by email address */
$profile = XN_Profile::load('email@address.com');

/* New: set attributes on profiles */
$profile->gender = XN_Profile::MALE;
$profile->gender = null;
$profile->birthday = '2005-12-10';
/* New: save a profile and its changes */
$profile->save();

/* For multiple profile load, all of the requested objects must be
 * screen names or all must be email addresses */
$profiles = XN_Profile::load(array('screen1', 'screen2', 'screen3'));
$profiles = XN_Profile::load(array('email1@domain.com', 'email2@domain.com', 'email3@domain.com'));
```

Use Cases First!

- What does the API **need** to do?
- (Not “what **could** it do?”)

Need-driven Development

- Adding is easy. Removing is hard.
- You have lots of freedom with arguments
- Accessors provide insulation

Arguments

Long parameter lists are toxic:

```
<?php
```

```
function save($data, $flavor = null, $scope = null,  
             $commit = null, $cascade = null,  
             $permissions = null) {  
    if (is_null($flavor))      { $flavor = 'quick'; }  
    if (is_null($scope))      { $scope = 'global'; }  
    if (is_null($commit))     { $commit = true; }  
    if (is_null($cascade))    { $cascade = false; }  
    if (is_null($permissions)) { $permissions = 0755; }  
    // ...  
}
```

What does this do?

```
<?php
```

```
save($data, null, null, true, false);
```

```
?>
```

Bread Stuffing



Bad Stuff



How about this?

```
<?php
```

```
save($data, array('scope' => 'local'));
```

```
?>
```

Ahh, much better:

```
<?php
```

```
function save($data, $paramsOrFlavor = null,  
             $scope = null, $commit = null,  
             $cascade = null, $permissions = null){  
    if (is_array($paramsOrFlavor)) {  
        // ...  
    }  
    else {  
        // ...  
    }  
}
```

Fun with `__get()` and `__set()`

```
public function __get($name) {
    switch ($name) {
        case self::screenName:
            return $this->_screenName;
        case self::fullName:
            return $this->_fullName;
        case self::uploadEmailAddress:
            $this->_lazyLoad('uploadEmailAddress');
            return $this->_uploadEmailAddress;
        case 'description':
            // ...
    }
}
```

Static 'n' Dynamic Analysis

- find + grep
- tokenizer
- hooks + logging

find + grep

```
find . -name \*.php -exec grep -H '::
```

- easy
- fast
- mostly correct: watch out for dynamic variable names, text collision, etc.

tokenizer

- php-specific knowledge, but...
- can be slower
- need to write custom rules and parsing

```
$tokens = array(T_INCLUDE => 0, T_INCLUDE_ONCE => 0,  
               T_REQUIRE => 0, T_REQUIRE_ONCE => 0);  
  
foreach (new PhpFilterIterator(new RecursiveIteratorIterator(  
    new RecursiveDirectoryIterator($root))) as $f) {  
    $muncher = new Tokenmunch(file_get_contents($f));  
    foreach ($muncher as $token) {  
        if (array_key_exists($token[0], $tokens)) {  
            $tokens[$token[0]]++;  
            $startOfLine = $muncher->scanBackFor(T_WHITESPACE, "/\n/");  
            $startOfBlock = $muncher->scanBackFor(T_OPEN_TAG);  
            $previousComment = $muncher->scanBackFor(T_COMMENT, "/\n$/");  
            $startPosition = max($startOfLine, $startOfBlock, $previousComment) + 1;  
            $endOfLine = $muncher->scanForwardFor(T_STRING, '/^;$/');  
            $slice = $muncher->sliceAsString($startPosition,  
                                             $endOfLine - $startPosition+1);  
  
            print trim($slice) . "\n";  
        }  
    }  
}
```

```
$x = $this->_buildPath('lib/helpers/Forum_Filter.php'); include_once $x;
require_once NF_APP_BASE . '/lib/XG_Embed.php';
$x = $this->_buildPath('lib/helpers/Forum_HtmlHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_CommentHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_Filter.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_CommentHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_SecurityHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_NotificationHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_NotificationHelper.php'); include_once $x;
require_once NF_APP_BASE . '/lib/XG_TagHelper.php';
require_once NF_APP_BASE . '/lib/XG_SecurityHelper.php';
$x = $this->_buildPath('lib/helpers/Forum_FileHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_SecurityHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_SecurityHelper.php'); include_once $x;
$x = W_Cache::getWidget('main')->findInclude('lib/helpers/Index_NotificationHelper.php'); require_once $x;
$x = $this->_buildPath('lib/helpers/Forum_NotificationHelper.php'); include_once $x;
require_once NF_APP_BASE . '/lib/XG_TagHelper.php';
require_once NF_APP_BASE . '/lib/XG_PaginationHelper.php';
$x = $this->_buildPath('lib/helpers/Forum_HtmlHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_SecurityHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_FileHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_CommentHelper.php'); include_once $x;
$x = W_Cache::getWidget('main')->findInclude('lib/helpers/Index_NotificationHelper.php'); require_once $x;
require_once NF_APP_BASE . '/lib/XG_FeedHelper.php';
require_once NF_APP_BASE . '/lib/XG_MetatagHelper.php';
$x = $this->_buildPath('lib/helpers/Forum_CommentHelper.php'); include_once $x;
require_once NF_APP_BASE . '/lib/XG_TagHelper.php';
require_once NF_APP_BASE . '/lib/XG_SecurityHelper.php';
$x = $this->_buildPath('lib/helpers/Forum_FileHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_SecurityHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_SecurityHelper.php'); include_once $x;
require_once NF_APP_BASE . '/lib/XG_SecurityHelper.php';
require_once NF_APP_BASE . '/lib/XG_FileHelper.php';
require_once NF_APP_BASE . '/lib/XG_HttpHelper.php';
require_once NF_APP_BASE . '/lib/XG_TagHelper.php';
$x = $this->_buildPath('lib/helpers/Forum_SecurityHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_HtmlHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_FileHelper.php'); include_once $x;
$x = $this->_buildPath('lib/helpers/Forum_UserHelper.php'); include_once $x;
require_once NF_APP_BASE . '/lib/XG_ActivityHelper.php';
```

Hooks + Logging

- need to instrument the API beforehand
- watch out for performance overhead

API for the API: XN_Event

```
class XN_Event {
    /**
     * Fire an event with optional arguments
     *
     * @param string $event
     * @param array $args optional arguments to pass to listeners
     */
    public static function fire($event, $args = null);

    /**
     * Listen for an event
     *
     * @param string $event
     * @param callback $callback Function to run when the event is fired
     * @param array $args optional arguments to pass to the callback
     * @return string
     */
    public static function listen($event, $callback, $args = null);
}
```

XN_Event in Use

XN_Content::save() calls:

```
XN_Event::fire('xn/content/save/before', array($this));  
// do the save  
XN_Event::fire('xn/content/save/after', array($this));
```

This has been very useful for cache expiration and selective runtime debugging instrumentation.

Late Static Binding Workaround

Class name registry for static inheritance:

```
W_Cache::putClass( 'app' , 'XG_App' );  
  
// ... time passes ...  
  
$className = W_Cache::getClass($role);  
$retval = call_user_func_array(  
    array($className, $method),  
    $args  
);
```

Names Matter

- Namespacing / collisions
- Versioning

Namespacing

- At Ning, “XN” means “hands off”
 - class names
 - property names
 - xml namespace prefixes

Versioning

- YourClass and YourClass2...sigh.
- Using `include_path` and `auto_prepend_file`
- Version number in REST URLs:

<http://app.ning.com/xn/atom/1.0/content/...>

Docblocks: Yay!

```
/** It's easy to generate human(-ish)
 * readable documentation from
 * docblocks. (@see PHPDocumentor,
 * @see doxygen)
 *
 * And the documentation is close
 * to the code.
 */
public function debugulator() {
}
```

Docblocks: Boo!

```
/** What about examples and tutorials
 *  and all of the other thing that
 *  are not method or class
 *  specific?
 *
 *  Is this documentation up to date
 *  with the @version of the code?
 */
public function rebigulator() {
}
```

Too Much Sugar

```
// "System" Attribute  
$content->title = 'Duck with Pea Shoots';  
  
// "Developer" Attribute  
$content->my->meat = true;
```

Non-literal Names = HFCS

```
$attrs = array('title', 'my->meat');  
  
foreach ($attrs as $attr) {  
    print "$attr is " . $content->$attr;  
}
```



Alternatives

OK `$content->title` and `$content->my_flavor;`

OK `$content->xn_title` and `$content->flavor;`

NO `$content['title']` and `$content->flavor;`

Testing & Code Coverage

The extent of your
test suite
is the
strength of your contract
with your users.

Tools are secondary.

Discipline is primary.

Tools

- SimpleTest



- http://www.lastcraft.com/simple_test.php

- PHPUnit

- <http://phpunit.de/>



To Keep in Mind...

- Lean towards use cases rather than unconstrained possibilities
- Naming, versioning, and documentation are not afterthoughts.
- Test suite code coverage is all you have to guarantee backwards-compatibility
- Sugar, yes; HFCS, no.

Resources

- Joshua Bloch: "How to Design a Good API and Why It Matters"
 - <http://lcsd05.cs.tamu.edu/slides/keynote.pdf>
- Zend Framework Documentation
 - <http://framework.zend.com/manual/manual/>
- eZ Components Documentation
 - <http://ez.no/doc/components/overview/>
- These slides: <http://www.sklar.com/blog/>

Come work at **Ning** !

- Write the code that powers 118,000+ social networks
- Write the next version of our PHP API
- Work in Palo Alto (or not)
- <http://jobs.ning.com> - david@ninginc.com