Declarative Development using Annotations in PHP

Frank Kleine & Stephan Schmidt

Agenda

- The Speakers
- Basic Concept of Annotations
- Annotations in Java
- Annotations in PHP
- Annotations in Stubbles
  - Usage examples
  - Defining your own annotations
- Q&A

Frank Kleine

- Working at 1&1 Internet Inc.
- PHP since 2000
- Stubbles Lead Developer
- Co-author of "Exploring PHP"

Stephan Schmidt

- Team Leader at 1&1 Internet Inc.
- Contributor to the PHP Open Source Community since 2001
- 15 PEAR packages, 1 pecl extension
- Author of „PHP Design Patters“ and co-author of several other PHP-related books
- Speaker at various conferences since 2001

The audience?

- Who is using PHP5?
- Who is using object-oriented development?
- Who is using PHP‘s reflection features?

Basic concept of Annotations

- Add metadata to classes, methods, properties
- Do not (directly) affect program semantics
- Can be used by tools or libraries
- Can be parameterized or simple marker annotations

Example scenarios

- Marking classes/methods as accessible via a web service
- Marking methods as unit test methods
- Defining how an object should be persisted or serialized
- Automating dependency injection

Annotations in Java 4

- Doclets, similar to PHPDoc
- Accessible via a Doclet API and a command line tool
- Not only used for documentation purposes
- Xdoclet for creation of EJBs and more

Annotations in Java 5

- Annotations are part of the language

```
public @interface MyAnnotation {
    String myParam();
}

@MyAnnotation(myParam="Foo")
public class MyClass {}
```

- Also accessible at runtime

Annotations in PHP

Annotations are not part of any PHP version.

Is this the end of the session?

Annotations in PHP (revisited)

- Probably 90% of you used some kind of annotation

```
/**
 * My class
 *
 * @author The Stubbles Team
 * @see http://www.stubbles.net
 */
class MyClass {}
```

History of annotations in PHP

1. There was PHPDoc
2. There was PHPDoc
3. PHP 5 was released
4. Some frameworks accessed PHPDoc tags
5. Some frameworks started using some "specialized" doc comments

Specialized frameworks

- Annotations embedded in DocBlocks
- Allow you to use annotations supported by the framework.
- No generic parser for annotations, but mostly some regexps
- Not (easily) possible to include new annotations

Extended Reflection API

- Provides access to PHPDoc comments
- Used to get information about the types of parameters or properties
- No real annotation support
- http://instantsvc.toolslave.net/wiki/ExtendedReflectionAPI

PHP_Unit

```
class Calculator {
    /**
     * @assert (0, 0) == 0
     * @assert (1, 0) == 1
     * @assert (1, 1) == 2
     */
    public function add($a, $b)
    {
        return $a + $b;
    }
}
```

SCA/SDO

```
class MyApplication  {

    /**
     * The stock quote service to use.
     *
     * @reference
     * @binding.wsdl  ../StockQuote/StockQuote.wsdl
     */
     public $stock_quote;
     ...
}
```

More specialized Frameworks

- Services_Webservice makes use of "pimped" PHPDoc comments to create WSDL
- EZPDO uses annotations to ease object-relational mapping

Next step: Generic frameworks

- Enable you to create your own annotations
- Provide generic annotation parsers
- Provide access to annotations at run-time
- No common standard for annotations

PEAR::PHP_Annotation

- Parser for generic DocBlocks and Java-style annotations
- Provides mechanism for dependency injection
- No public release, yet
- First and last commit 4 months ago

Addendum

- Supports Java-style annotations
- Annotations are classes that extend class **`Annotation`**
- Annotations are accessible via reflection classes that extend built-in reflection
- http://code.google.com/p/addendum/

Stubbles

- OOP PHP 5.2 framework
- Package based
- Provides generic annotation functionality
- Eats its own dog food and uses annotations in various packages

Annotations in Stubbles

- Persistence
  - Serializing objects to XML
  - Persisting objects in RDMBS
- Security
  - Marking methods as callable via JSON-RPC
  - Marking methods callable from XSLT
- Misc
  - Managing dependency injection

A Person POPO

```
class Person {
    protected $id;
    protected $name;
    protected $age;
    protected $role;

    public function __construct($id, $name, $age, $role = 'user') {
        $this->id   = $id;
        $this->name = $name;
        $this->age  = $age;
        $this->role = $role;
    }

    public function getId() {
        return $this->id;
    }
    ... more getter methods for the other properties ...
}
```

Serializing to XML

```
// Create a serializer
$serializer = new stubXMLSerializer();

// XML in Stubbles is created via a streaming API
$writer = stubXMLStreamWriterFactory::createAsAvailable();

// Create and serialize the object
$user = new Person (1, 'schst', 33, 'admin');
$serializer->serialize($user, $writer);

echo writer->asXML();
```

The resulting XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Person>
  <getId>1</getId>
  <getName>schst</getName>
  <getAge>33</getAge>
  <getRole>admin</getRole>
</Person>
```

XMLSerializer behaviour

- All public methods without parameters are exported
- Class and method names are used as tag names
- But: The behavior can be changed by annotating the class.

XMLSerializer annotations

Changing the name of the root tag:

```
/**
 * A person
 *
 * @XMLTag(tagName="user")
 */
class Person {
    ... rest of the code ...
}
```

XMLSerializer annotations

Changing the name of other tags

```
/**
 * Get the name
 *
 * @XMLTag(tagName="realname")
 * @return string
 */
public function getName() {
    return $this->name;
}
```

XMLSerializer annotations

Using attributes instead of tags

```
/**
 * Get the id
 *
 * @XMLAttribute(attributeName="userId")
 * @return int
 */
public function getId() {
    return $this->id;
}
```

XMLSerializer annotations

Ignoring methods

```
/**
 * Get the age
 *
 * @XMLIgnore
 * @return int
 */
public function getAge() {
    return $this->age;
}
```

The modified XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<user userId="1">
  <realname>schst</realname>
  <role>admin</role>
</user>
```

- tag names have been changed
- id property is serialized as attribute
- age is omitted

Features of XMLSerializer

- Serializes (nearly) every data structure
- Able to use ext/dom or ext/xmlwriter
- Able to serialize public methods and properties
- Provides annotations to "batch-process" methods and properties

Persistence: POPO revisited

```
/**
 * @DBTable(name='persons')
 */
class Person extends stubAbstractPersistable {
    ... properties like in old version ...
    /**
     * @DBColumn(name='person_id', isPrimaryKey=true)
     */
    public function getId() { return $this->id; }
    }
    /**
     * @DBColumn(name='name', defaultValue='')
     */
    public function getName() { return $this->name; }
    /**
     * @DBColumn(name='age', defaultValue=0)
     */
    public function getAge() { return $this->age; }
    ... more getter and of course setter methods for the other properties ...
}
```

Persistence: insert

```
... create the person with the data ...
$person = new Person();
$person->setName('Frank Kleine');
$person->setAge(27);
$person->setRole('admin');

... get the connection ...
$connection = stubDatabaseConnectionPool::getConnection();
... get the serializer ...
$serializer = stubDatabaseSerializer::getInstance($connection);
... and save to database ...
$serializer->serialize($person);
... done: ...
var_dump($person->getId());
... displays int(1) ...
```

Persistence: select (find)

```
... create the person object ...
$person = new Person();
$person->setId(1);
... get the connection ...
$connection = stubDatabaseConnectionPool::getConnection();
... get the finder ...
$finder = stubDatabaseFinder::getInstance($connection);
... and retrieve data ...
$finder->findByPrimaryKeys($person);
... done: ...
var_dump($person->getAge());
... displays int(27) ...

or:
$criterion = new stubEqualCriterion('age', 27);
$persons = $finder->findByCriterion($criterion, 'Person');
... $persons is now an array containing a list of Person objects all of age 27
```

Persistence: more features

- Delete similarly to finder
- Update works like insert
- Very simple support for joins (needs more research)
- Independent of database: connection handle knows database type, correct query builder is selected

Persistence: going crazy

```
/**
 * @DBTable(name='persons', type='InnoDB')
 */
class Person extends stubAbstractPersistable {
    ... properties like in old version ...
    /**
     * @DBColumn(name='person_id', isPrimaryKey=true, type='int', size=10,
    isUnsigned=true)
     */
    public function getId() { return $this->id; }
    }
    /**
     * @DBColumn(name='name', defaultValue='', type='varchar', size=255,
    isNullable=fals)
     */
    public function getName() { return $this->name; }
 }
$connection = stubDatabaseConnectionPool::getConnection();
$creator = stubDatabaseCreator::getInstance($connection);
$creator->createTable('Person');
```

Accessing annotations in PHP

PHP's built-in reflection API:

```
$class = new ReflectionClass('Person');
echo $class->getName() . "\n";
foreach ($class->getMethods() as $method) {
    echo " -> " . $method->getName()  . "\n";
}
Person
 -> __construct
 -> getId
 -> getName
 -> ...
```

Accessing annotations in PHP

Stubbles' reflection API:

```
$class = new stubReflectionClass('Person');
if ($class->hasAnnotation('XMLTag')) {
   $xmlTag = $class->getAnnotation('XMLTag');
   print_r($xmlTag);
}

stubXMLTagAnnotation Object
(
    [tagName:protected] => user
    [elementTagName:protected] =>
    [annotationName:protected] => XMLTag
)
```

Stubbles' reflection

- Provides drop-in replacements for all built-in reflection classes, that extend the built-in classes (for type safety)
- Provides two additional methods:
  - `hasAnnotation(string name)`
  - `getAnnotation(string name)`
- Annotations are objects

Creating annotations

- Annotation classes must implement `stubAnnotation` interface
- Annotations can extend abstract base class
- Annotations should comply to naming scheme
- Annotations may contain additional properties and methods

Example: CSV Export

```
/**
 * A person
 *
 * @CSV(file="users.csv",
 *      delimeter=";")
 */
class Person {
    ...properties and methods ...
    /**
     * Get the id
     *
     * @CSVField
     * @return int
     */
    public function getId() {
        return $this->id;
    }
}
```

The @csv Annotation

```
class stubCSVAnnotation
    extends stubAbstractAnnotation
    implements stubAnnotation {
}
```

Annotation parameters

- Parameters can be set in three ways
  - Providing a public property with the same name as the parameter
  - Providing a public method with the same name as the parameter prefixed with `set`
  - Providing a magic `__set()` method

The @csv Annotation

```
class stubCSVAnnotation
    extends stubAbstractAnnotation
    implements stubAnnotation {
    public $file;
    protected $delimeter;
    public function setDelimeter($delim) {
        $this->delimeter = $delim;
    }
    public function getDelimeter() {
        return $this->delimeter;
    }
}
```

Annotation parameter types

- Strings (enclosed in " or ')
- integers and doubles
- **`true`**, **`false`** and **`null`**
- PHP constants for configurable annotations
- instances of stubReflectionClass (**`clazz=Person.class`**)

Annotation targets

- Annotations may be added to
  - Classes
  - Properties
  - Methods
  - Functions
- Each annotation may restrict the possible targets

The @csv Annotation

- Should only be added to classes

```
class stubCSVAnnotation extends stubAbstractAnnotation
   implements stubAnnotation {
     public function getAnnotationTarget() {
         return stubAnnotation::TARGET_CLASS;
     }
}
```

- Parser will throw an exception on error

The `@CSVField` Annotation

- Can be added to methods and properties

```
class stubCSVFieldAnnotation
    extends stubAbstractAnnotation
    implements stubAnnotation {
    public function getAnnotationTarget() {
        return stubAnnotation::TARGET_METHOD
                | stubAnnotation::TARGET_PROPERTY;
    }
}
```

Implementing the CSV Writer

1. Accept any object
2. Check for `@CSV` annotation
3. Extract parameters from annotation
4. Traverse all methods and properties
5. Check for `@CSVField` annotation
6. Invoke the methods
7. Write line to file

The `CSVWriter` class

```
class CSVWriter {
    public static function write($obj) {
        $class = new stubReflectionClass(get_class($obj));
        if (!$class->hasAnnotation('CSV')) {
            throw new Exception('Not suported');
        }
        $csv = $class->getAnnotation('CSV');
        $fp = fopen($csv->file, 'a');
        $fields = array();
        foreach ($class->getMethods() as $method) {
```

The `CSVWriter` class

```
            if (!$method->hasAnnotation('CSVField')) {
                continue;
            }
            $fields[] = $method->invoke($obj);
        }
        $line = implode($csv->getDelimeter(), $fields);
        fwrite($fp, $line . "\n");
        fclose($fp);
    }
}
```

Using the `CSVWriter` class

- Pass any annotated object to the `write()` method:

```
$user = new Person(1, 'schst', 33, 'admin');
CSVWriter::write($user);

users.csv:
1;schst;33
```

Advances Features

- Annotations are cached during and between the requests
- Annotations can be modified at run-tim

Advanced features

Annotations may be interfaces that can be implemented in various ways:

```
/**
 * @XMLMethods[XMLMatcher](pattern='/^get(.+)/')
 */
class Person {
    ... methods ...
}
```

Annotation interfaces

- XMLSerializer only knows about the `@XMLMethods` annotation interface
- `XMLMatcher` implements this interface and decides which methods to export
- Annotations do not only provide meta information, but also logic

The end

Thank you for your attention.

Want to know more?

http://www.stubbles.net

http://www.frankkleine.de

http://www.schst.net

Commercial break