Zend_Json_Jsonable - Jurrien Stutterheim

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

<ac:macro ac:name="unmigrated-inline-wiki-markup"><ac:plain-text-body><![CDATA[

Zend Framework: Zend_Json_Jsonable Component Proposal

Proposed Component Name	Zend_Json_Jsonable
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Json_Jsonable
Proposers	Jurriën Stutterheim
Zend Liaison	Matthew Weier O'Phinney
Revision	1.0 - 21 August 2008: Initial Draft. (wiki revision: 10)

Table of Contents

1. Overview
2. References
3. Component Requirements, Constraints, and Acceptance Criteria
4. Dependencies on Other Framework Components
5. Theory of Operation
6. Milestones / Tasks
7. Class Index
8. Use Cases
9. Class Skeletons

1. Overview

Zend_Json_Jsonable is a simple interface that provides a toJson method. This allows for a more fine-grained control over how objects serialize to JSON.

2. References

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **will** provide an interface that allows more control over serializing objects to JSON
- This component **will** be supported by Zend_Json's encode facilities
- This component **will not** be supported by PHP's native json_encode() function (yet).

4. Dependencies on Other Framework Components

- Zend_Json

5. Theory of Operation

Currently object support for JSON encoding is very limited – only public attributes can be serialized. The Zend_Json_Jsonable is designed to work around this limitation. The __toJson() method that's provided with this interface can be used by Zend_Json's encode methods to retrieve a custom set of data from the object.

A question for this proposal

- Should the __toJson() method be prefixed with __?

6. Milestones / Tasks

- Milestone 1: **[DONE]** Initial proposal
- Milestone 2: Receive and process comments
- Milestone 3: Component is incubated
- Milestone 4: Zend_Json is updated to support this interface
- Milestone 5: Documentation is available.
- Milestone 6: Component is included in the core.

7. Class Index

- Zend_Json_Jsonable

8. Use Cases

UC-01: Zend_Paginator and ContextSwitch

```
// Zend_Paginator implements Zend_Json_Jsonable, so it can be handled better by the ContexSwitch
class Zend_Paginator implements Countable, IteratorAggregate, Zend_Json_Jsonable
{
    public function __toJson();
    {
        return $this->getCurrentItems();
    }
}

// With the interface in place, this can be shortened...
public function listAction()
{
    $entries = new Entries();
    $paginator = $entries->getPaginated();

    $paginator->setCurrentPageNumber($this->_getParam('page', 1));

    if ($this->getRequest()->isXmlHttpRequest()) {
        $this->view->entries = $paginator->getCurrentItems();
    } else {
        $this->view->entries = $paginator;
    }
}

// ... to this
public function listAction()
{
    $entries = new Entries();
    $paginator = $entries->getPaginated();

    $paginator->setCurrentPageNumber($this->_getParam('page', 1));

    $this->view->entries = $paginator;
}
```

9. Class Skeletons

```
interface Zend_Json_Jsonable
{
    /**
     * Return an array which will be serialized as JSON
     *
     * @return array|string
     */
    public function __toJson();
}
```

```
// Proposed change to Zend_Json_Encoder
/**
 * Encode an object to JSON by encoding each of the public properties
 *
 * A special property is added to the JSON object called '__className'
 * that contains the name of the class of $value. This is used to decode
 * the object on the client into a specific class.
 *
 * @param $value object
 * @return string
 * @throws Zend_Json_Exception If recursive checks are enabled and the object has been serialized
previously
 */
protected function _encodeObject(&$value)
{
    if ($this->_cycleCheck) {
        if ($this->_wasVisited($value)) {
            throw new Zend_Json_Exception(
                'Cycles not supported in JSON encoding, cycle introduced by '
                . 'class "' . get_class($value) . '"'
            );
        }

        $this->_visited[] = $value;
    }

    if ($value instanceof Zend_Json_Jsonable) {
        $props = $this->_encodeArray($value->__toJson());
    } else {
        $props = '';
        foreach (get_object_vars($value) as $name => $propValue) {
            if (isset($propValue)) {
                $props .= ','
                        . $this->_encodeValue($name)
                        . ':'
                        . $this->_encodeValue($propValue);
            }
        }
    }

    return '{"__className":"' . get_class($value) . '"'
            . $props . '}';
}
```

```
// Proposed change to Zend_Json::encode();
/**
 * Encode the mixed $valueToEncode into the JSON format
 *
 * Encodes using ext/json's json_encode() if available.
 *
 * NOTE: Object should not contain cycles; the JSON format
 * does not allow object reference.
 *
 * NOTE: Only public variables will be encoded
 *
 * @param mixed $valueToEncode
 * @param boolean $cycleCheck Optional; whether or not to check for object recursion; off by default
 * @return string JSON encoded object
 */
public static function encode($valueToEncode, $cycleCheck = false)
{
    if (function_exists('json_encode') && self::$useBuiltinEncoderDecoder !== true) {
        if ($valueToEncode instanceof Zend_Json_Jsonable) {
            $valueToEncode = $valueToEncode->__toJson();
        }

        return json_encode($valueToEncode);
    }

    require_once 'Zend/Json/Encoder.php';
    return Zend_Json_Encoder::encode($valueToEncode, $cycleCheck);
}
```

]]></ac:plain-text-body></ac:macro>
]]></ac:plain-text-body></ac:macro>