Zend_Soap_SimpleTypeValuesValidator - Jeannie Boffel

<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_Soap_SimpleTypeValuesValidator Component Proposal

Proposed Component Name	Zend_Soap_SimpleTypeValuesValidator
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Soap_SimpleTypeValuesValidator
Proposers	Jeannie BOFFEL
Zend Liaison	TBD
Revision	1.0 - 07 December 2010: Initial Draft. (wiki revision: 9)

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

I'd like to introduce to Zend Framework a new layer to check/validates values returned in a SOAP response against what user declared in the schema through auto discovery system.

2. References

- Zend Framework AutoDiscovery
- ZF-10789
- Zend_Soap_ComplexTypeValuesValidator

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **will** add support for checking validity of values against XSD declaration from auto discovery support.

- This component **will not** make sure values are meaningful.
- This component **will not** validate the XSD generated. Meanning we don't prevent user from doing wrong XSD by adding unexisting attribute to an element for example.
- This component **will not** be exhaustive and **will not** support all XSD features immediately.

4. Dependencies on Other Framework Components

- PHP >= 5.3.0
- Zend_Soap
- Zend_Soap_AutoDiscovery
- Zend_Soap_Wsdl_Strategy_DefaultComplexType (modified by proposal)

5. Theory of Operation

The component is instantiated through extending of class.

6. Milestones / Tasks

- Feature Request 1: Support more XSD base type than just PHP variable base set type.

- Milestone #: [DONE] Design
- Milestone #: [DONE] Proof of concept by implement new complex type and simple type value checking
- Milestone #: [DONE] Start support of all base type compatible with PHP variable base set type
- Milestone #: [DONE] Start support of user defined type through SimpleType in XSD
- Milestone 1: Working prototype checked into the incubator supporting use cases #1.
- Milestone 2: Unit tests exist, work, and are checked into SVN.
- Milestone 3: Initial documentation exists.

7. Class Index

- Zend_Soap_SimpleTypeValuesValidator

8. Use Cases

UC-01

Definition of one simple type and call its extended test method.

```
/**
 * @xsd simpleType
 * @xsd restriction array('base' => 'xsd:string')
 */
class FilterTypeString extends Zend_Soap_SimpleTypeValuesValidator
{
 /**
  * @xsd enumeration array('value' => 'test1')
  * @xsd enumeration array('value' => 'test2')
  * @xsd enumeration array('value' => 'test3')
  */
 public $restriction;
}

var_dump(FilterTypeString::getInstance()->test('test1'));
var_dump(FilterTypeString::getInstance()->test('anything'));
```

Output should be like:
boolean true
boolean false

UC-02

Again with an union of simple type.

```
/**
 * @xsd simpleType
 * @xsd restriction array('base' => 'xsd:string')
 */
class FilterTypeString extends Zend_Soap_SimpleTypeValuesValidator
{
 /**
  * @xsd enumeration array('value' => 'test1')
  * @xsd enumeration array('value' => 'test2')
  * @xsd enumeration array('value' => 'test3')
  */
 public $restriction;
}
/**
 * @xsd simpleType
 * @xsd restriction array('base' => 'xsd:integer')
 */
class FilterTypeInteger extends Zend_Soap_SimpleTypeValuesValidator
{
 /**
  * @xsd enumeration array('value' => '1')
  * @xsd enumeration array('value' => '2')
  * @xsd enumeration array('value' => '3')
  */
 public $restriction;
}
/**
 * @xsd simpleType
 * @xsd union
 */
class FilterType extends Zend_Soap_SimpleTypeValuesValidator
{
 /**
  * @xsd simpleType FilterTypeString
  * @xsd simpleType FilterTypeInteger
  */
 public $union;
}

var_dump(FilterType::getInstance()->test(1));
var_dump(FilterType::getInstance()->test('test1'));
var_dump(FilterType::getInstance()->test('anything'));
```

Output should be like:
boolean true
boolean true
boolean false

UC-03

Again with a list of simple type (using previous union definition too).

```
/**
 * @xsd simpleType
 * @xsd list
 */
class FilterListType extends Zend_Soap_SimpleTypeValuesValidator
{
 /**
  * @xsd itemType FilterType
  */
 public $list;
}

var_dump(FilterTypeString::getInstance()->test('test1 1
test2'));
var_dump(FilterTypeString::getInstance()->test('anything else'));
```

Output should be like:
boolean true
boolean false

9. Class Skeletons

```
class Zend_Soap_SimpleTypeValuesValidator {
    /*
     * Keep memory of children class name for
     * easy use.
     */
    private $_className;
    /*
     * Result of reflection on children class name
     * is stored here. We do no wish to compute
     * reflection each time.
     */
    private $_classReflection;
    /*
     * List of check to do for a final type
     */
    private $_checkMap;
    /*
     * Memory of is simple type a restriction,
     * union or list.
     */
    private $_simpleTypeBase = null;
    /*
     * List of all uniq instance of children
     * already instanciated.
     */
    private static $_uniqInstance;

    /**
     * Function to call to launch the test on value
     *
     * @param   mixed       $value
     * @return  bool        true on success
     * @throws  SoapFault   on error
     */
    public function test($value) {

        if($this->_simpleTypeBase === null) {
            if (preg_match_all('/@xsd\s+([^\s]+)([ \t\f]+(.*))?/m',
```

```
$this->_classReflection->getDocComment(), $firstMatches)) {
                foreach ($firstMatches[0] as $firstKey => $someValue) {
                    switch ($firstMatches[1][$firstKey]) {
                        case 'restriction' :
                            $this->_simpleTypeBase = 'restriction';
                            //$firstMatches[2][$firstKey] = substr($firstMatches[2][$firstKey], 0,
-1);
                            //TODO: have to think about how it could be implemented to match value
                            //      of restriction against another simpletype in base property
                            //$this->_addAttributes($firstMatches[2][$firstKey], $all);
                            foreach ($this->_classReflection->getProperties() as $property) {
                                if ($property->isPublic()
                                     && preg_match_all('/@xsd\s+([^\s]+)([ \t\f]+(.*))?/m',
$property->getDocComment(), $matches)) {
                                     $this->_compileCheckMap($matches);
                                }
                            }
                            break;
                        case 'union' :
                            $this->_simpleTypeBase = 'union';
                            //$firstMatches[2][$firstKey] = substr($firstMatches[2][$firstKey], 0,
-1);
                            foreach ($this->_classReflection->getProperties() as $property) {
                                if ($property->isPublic()
                                     && preg_match_all('/@xsd\s+simpleType([ \t\f]+(.*))?/m',
$property->getDocComment(), $matches)) {
                                    foreach ($matches[0] as $key => $someValue) {
                                        $matches[1][$key] = substr($matches[1][$key], 0, -1);
                                        if (class_exists($matches[1][$key])) {
                                            $this->_checkMap[0][] = 'simpleType';
                                            if (is_callable($matches[1][$key] . '::getInstance')) {
                                                $this->_checkMap[1][] = $matches[1][$key];
                                            } else {
                                                throw new Exception($matches[1][$key] . ' does not
extend SimpleTypeValuesValidator', 500);
                                            }
                                        } else {
                                            $this->_checkMap[0][] = 'baseType';
                                            $this->_checkMap[1][] = '';
                                        }
                                    }
                                }
                            }
                            break;
                        case 'list' :
                            $this->_simpleTypeBase = 'list';
                            //$firstMatches[2][$firstKey] = substr($firstMatches[2][$firstKey], 0,
-1);
                            foreach ($this->_classReflection->getProperties() as $property) {
                                if ($property->isPublic()
                                     && preg_match_all('/@xsd\s+itemType([ \t\f]+(.*))?/m',
$property->getDocComment(), $matches)) {
                                    $this->_checkMap[0][] = 'itemType';
                                    foreach ($matches[0] as $key => $someValue) {
                                        $matches[1][$key] = substr($matches[1][$key], 0, -1);
                                        if (class_exists($matches[1][$key])) {
                                            if (is_callable($matches[1][$key] . '::getInstance')) {
                                                $this->_checkMap[1][] = $matches[1][$key];
                                            } else {
                                                throw new Exception($matches[1][$key] . ' does not
extend SimpleTypeValuesValidator', 500);
                                            }
                                        } else {
                                            throw new Exception($matches[1][$key] . ' does not exists
as class', 500);
                                        }
                                    }
                                }
                            }
                            break;
```

```
                }
            }
        }
    }

    switch ($this->_simpleTypeBase) {
        case 'restriction' :
            return self::_checkValues($this->_checkMap[0], $this->_checkMap[1], $value);
            break;
        case 'union' :
            foreach ($this->_checkMap[0] as $key => $simpeTypeElement) {
                if ($simpeTypeElement == 'simpleType') {
                        $className = $this->_checkMap[1][$key];
                        $result = $className::getInstance()->test($value);
                        if ($result === true)
                            return true;
                } else {
                    if(self::checkBaseType($matches[1][$key], $value))
                        return true;
                }
            }
            return false;
            break;
        case 'list' :
            foreach ($this->_checkMap[0] as $key => $simpeTypeElement) {
                foreach ($this->_checkMap[1] as $listKey => $listType) {
                    if (preg_match_all('/[^\s]+/m', $value, $listMatches)) {
                        foreach ($listMatches[0] as $listValueKey => $listValue) {
                            $result = $listType::getInstance()->test($listValue);
                            if ($result !== true)
                                return false;
                        }
                        return true;
                    }
                }
            }
            return false;
            break;
    }

    return false;
}

/**
 * Function to get the wanted children instance.
 *
 * To get it use children_class_name::getInstance()
 * To launch a test: children_class_name::getInstance()->test(some_value)
 * Chilren must extends SimpleTypeValuesValidator class.
 *
 * @return  object
 */
public static function getInstance() {
    $className = get_called_class();
    if (!isset(self::$_uniqInstance[$className])) {
        self::$_uniqInstance[$className] = new $className;
        self::$_uniqInstance[$className]->_className = $className;
        self::$_uniqInstance[$className]->_classReflection = new ReflectionClass($className);
    }
    return self::$_uniqInstance[$className];
}

/**
 * Function to prepare the map to execute all needed check.
 *
 * @param array $matches
 */
protected function _compileCheckMap($matches) {

    $this->_checkMap[0] = $matches[1];
```

```
        foreach ($matches[0] as $key => $someValue) {
            $matches[2][$key] = substr($matches[2][$key], 0, -1);
            eval('$this->_checkMap[1][$key] = ' . $matches[2][$key] . ';');
        }
    }

    /**
     * Function to check all final value against declared type.
     *
     * @param array $matches
     * @param array $matchesArgs
     * @param mixed $value
     */
    protected static function _checkValues($matches, $matchesArgs, $value) {
        foreach ($matches as $key => $checkType) {
            switch ($checkType) {
                case 'minExclusive' : return true;
                    if ($value >= $matchesArgs[$key]['value'])
                        return false;
                    break;
                case 'minInclusive' :
                    if ($value > $matchesArgs[$key]['value'])
                        return false;
                    break;
                case 'maxExclusive' :
                    if ($value <= $matchesArgs[$key]['value'])
                        return false;
                    break;
                case 'maxInclusive' :
                    if ($value < $matchesArgs[$key]['value'])
                        return false;
                    break;
                case 'totalDigits' :
                    if (strlen($value) != $matchesArgs[$key]['value'])
                        return false;
                    break;
                case 'fractionDigits' :
                    if (strlen(strchr('.', $value))-1 != $matchesArgs[$key]['value'])
                        return false;
                    break;
                case 'length' :
                    if (strlen($value) != $matchesArgs[$key]['value'])
                        return false;
                    break;
                case 'minLength' :
                    if (strlen($value) < $matchesArgs[$key]['value'])
                        return false;
                    break;
                case 'maxLength' :
                    if (strlen($value) > $matchesArgs[$key]['value'])
                        return false;
                    break;
                case 'enumeration' :
                    if ($matchesArgs[$key]['value'] == $value)
                        return true;
                    break;
                case 'whiteSpace' :
                    // dixit W3C documentation, no validation rules for that.
                    return true;
                    break;
                case 'pattern' :
                    if (!preg_match("/" . $matchesArgs[$key]['value'] . "/", $value))
                        return false;
                    break;
                default :
                    throw new Exception($checkType . ' unknown SimpleTypeValuesValidator', 500);
            }
        }
        if ($checkType == 'enumeration')
            return false;
```

```
        else
            return true;
    }

    /**
     * Function to check if a value is part of
     * base type of any XSD
     *
     * @param string $baseType
     * @param mixed $value
     */
    public static function checkBaseType($baseType, $value) {
        switch ($baseType) {
            case 'string':
            case 'str':
                return is_string($value);
            case 'int':
            case 'integer':
                return is_int($value);
            case 'float':
            case 'double':
                return is_float($value);
            case 'boolean':
            case 'bool':
                return is_bool($value);
            case 'array':
                return is_array($value);
            case 'object':
                return is_object($value);
            case 'mixed':
                return true;
            case 'void':
                return empty($value);
            default:
                throw new Exception($baseType . ' unknown SimpleTypeValuesValidator BaseType', 500);
```

```
            }
        }
    }
```

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