Zend_Soap_Wsdl_Strategy_DefaultComplexType - Improvement for the AutoDiscover - 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_Wsdl_Strategy_DefaultComplexType Component Proposal

Proposed Component Name	Zend_Soap_Wsdl_Strategy_DefaultComplexType
Developer Notes	http://framework.zend.com/wiki/display/ZFDEV/Zend_Soap_Wsdl_Strategy_DefaultComplexType
Proposers	Jeannie BOFFEL
Zend Liaison	TBD
Revision	1.0 - 11 November 2010: Initial Draft. (wiki revision: 30)

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

Improvement of Zend_Soap_Wsdl_Strategy_DefaultComplexType goal is to give a real possibility to describe XSD for auto-generated WSDL via PHP Document in commentaries without breaking existing code using current version.

The goal is not to give a full support of XSD technology. But at least, we could try to make possible usage of 80% or 90% of basic component of XSD. Most used one. I think this goal is reach with current version.

2. References

- Zend Framework AutoDiscovery

3. Component Requirements, Constraints, and Acceptance Criteria

- This component **will** add support for describing XSD of auto-generated WSDL through the use of the AutoDiscovery.
- This component **will** replace existing Zend_Soap_Wsdl_Strategy_DefaultComplexType component.
- This component **will** generate same XSD+WSDL than previous component if new features are not used (not breaking code).
- This component **will** reserve php Doc tag @xsd for any action linked to new features.
- 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

- Zend_Soap

5. Theory of Operation

The component is instantiated by AutoDiscovery system of Zend Framework when trying to auto-generate a WSDL+XSD via reflection on the code.

6. Milestones / Tasks

- Feature Request 1: I tried to add support for use ref attributes in element tag,
  but the generated soap response with soap server is not compatible for now.
  WSDL+XSD are correct, but response will not match the auto-generated XSD...
  So use only complexType with call to them with style attribute.
- Feature Request 2: Add support for SOA design, meaning improve again AutoDiscovery, improve also Zend_Soap_Wsdl which has a bug in its way to genrate the XML, improve Zend_Soap_Server because register exception system is not really usable as is with idea of SOA design for example. However, it's also possible to decide to not use this support for now.
  I implemented already almost everything needed, except error handler, not yet sure about the design.

- Milestone #: [DONE] Design
- Milestone #: [DONE] Proof of concept by implement new complex type strategy
- Milestone #: [DONE] Start support of based most use tag of XSD, such as sequence, choice, all
- Milestone #: [DONE] Start support of basical restriction such as minOccurs, maxOccurs
- Milestone #: [DONE] Start support of simpleType through new class, like for complexType
- Milestone #: [DONE] Start support of more complex restriction type such as pattern, enumeration through simpleType support
- Milestone #: [DONE] Start support of union and list through simpleType support
- 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_Wsdl_Strategy_DefaultComplexType

8. Use Cases

To test validation of schema against xml response from a PHP SoapServer, for example with SoapUI, please generate in literal mode. Do not try to generate in document style instead of RPC.

UC-01

Basic exemple with 2 simpleType, 3 complexType, 1 function

```
class wsGetUserInfos
{
 /**
     * @param string $name
     * @return wsGetUserInfosResult
     */
 public function getUserInfos($name) {
  $details=new detailsBrotherList();
  $details->addBrotherDetail(new brotherDetail("Jean", 1, 10));
  $details->addBrotherDetail(new brotherDetail("Jack", 2, 20));
  $details->addBrotherDetail(new brotherDetail("Martin", 3, 30));

  $tmp = new wsGetUserInfosResult("Etienne", "Dupont", 33, $details);

  return objectToArray($tmp);

 }
}

function objectToArray($d) {
 if (is_object($d)) {
  // Gets the properties of the given object
  // with get_object_vars function
  $d = get_object_vars($d);
 }
 if (is_array($d)) {
  /*
  * Return array converted to object
  * Using __FUNCTION__ (Magic constant)
  * for recursive call
  */
  return array_map(__FUNCTION__, $d);
 }
 else {
  // Return array
  return $d;
 }
}

/**
 * @xsd sequence
 */
class wsGetUserInfosResult
{
 /**
  * @xsd sequence start
     * @var string
     */
 public $firstName;
 /**
  * @var string
  */
 public $familyName;
 /**
  * @var AgeType
  */
 public $age;
 /**
     * @var detailsBrotherList
     * @xsd sequence end
     */
 public $detailsBrotherList;

 /**
```

```
     * @param string $firstName
     * @param string $familyName
     * @param integer $age
     * @param detailsBrotherList $detailsBrotherList
     * @return wsGetUserInfosResult
     */
 public function __construct($firstName, $familyName, $age, $detailsBrotherList) {
  $this->firstName = $firstName;
  $this->familyName = $familyName;
  $this->age = $age;
  $this->detailsBrotherList = $detailsBrotherList;

  return $this;
 }
}

/**
 * @xsd simpleType array('id' => 1)
 * @xsd restriction array('base' => 'xsd:string')
 */
class AgeType
{
 /**
  * @xsd pattern array('value' => '[0-9][0-9][0-9]')
  */
 public $pattern;
}

/**
 * @xsd simpleType
 * @xsd restriction array('base' => 'xsd:string')
 */
class RankType
{
 /**
  * @xsd enumeration array('value' => '1')
  * @xsd enumeration array('value' => '2')
  * @xsd enumeration array('value' => '3')
  */
 public $enumeration;
}

/**
 * @xsd sequence
 * @xsd complexType
 */
class detailsBrotherList
{
 /**
     * @var brotherDetail array('minOccurs'=>0, 'maxOccurs'=>"unbounded")
     */
 public $brotherDetail;

 /**
     * @return detailsBrotherList
     */
 public function __construct() {
  $this->brotherDetail = array();
  return $this;
 }

 /**
     * @param brotherDetail $brotherDetail
     */
 public function addBrotherDetail($brotherDetail) {
  $this->brotherDetail[] = $brotherDetail;
 }

}
```

```
/**
 * @xsd sequence array('id' => 3)
 * @xsd complexType array('id' => 2)
 */
class brotherDetail
{
 /**
     * @var string
     */
 public $firstName;
 /**
  * @var RankType
  */
 public $rank;
 /**
     * @var AgeType
     */
 public $age;

 /**
     * @param string $firstName
     * @param RankType $rank
     * @param AgeType $age
     * @return brotherDetail
     */
 public function __construct($firstName, $rank, $age) {
  $this->firstName = $firstName;
  $this->rank = $rank;
  $this->age = $age;

  return $this;
```

```
 }

}
```

UC-02

Example of union of simpleType :

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

/**
 * @xsd simpleType
 * @xsd restriction array('base' => 'xsd:positiveInteger')
 */
class FilterTypeInteger
{
 /**
  * @xsd enumeration array('value' => '1')
  * @xsd enumeration array('value' => '2')
  * @xsd enumeration array('value' => '3')
  */
 public $enumeration;
}

/**
 * @xsd simpleType array('id'=>10)
 * @xsd union array('id'=>11)
 */
class FilterType
{
 /**
  * @xsd simpleType FilterTypeString
  * @xsd simpleType FilterTypeInteger
  */
 public $union;
}

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

Generated WSDL+XSD for Use Case 1

```
<?xml version="1.0"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://www.zend.com/getuserinfo"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap-enc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
name="wsGetUserInfos" targetNamespace="http://www.zend.com/getuserinfo">
      <types>
            <xsd:schema targetNamespace="http://www.zend.com/getuserinfo">
                  <xsd:simpleType name="AgeType" id="1">
                        <xsd:restriction base="xsd:string">
                              <xsd:pattern value="[0-9][0-9][0-9]"/>
                        </xsd:restriction>
                  </xsd:simpleType>
                  <xsd:simpleType name="RankType">
                        <xsd:restriction base="xsd:string">
                              <xsd:enumeration value="1"/>
                              <xsd:enumeration value="2"/>
                              <xsd:enumeration value="3"/>
                        </xsd:restriction>
                  </xsd:simpleType>
                  <xsd:complexType name="brotherDetail" id="2">
                        <xsd:sequence id="3">
                              <xsd:element name="firstName" type="xsd:string"/>
                              <xsd:element name="rank" type="tns:RankType"/>
                              <xsd:element name="age" type="tns:AgeType"/>
                        </xsd:sequence>
                  </xsd:complexType>
                  <xsd:complexType name="detailsBrotherList">
                        <xsd:sequence>
                              <xsd:element ref="tns:brotherDetail" minOccurs="0"
maxOccurs="unbounded"/>
                        </xsd:sequence>
                  </xsd:complexType>
                  <xsd:complexType name="wsGetUserInfosResult">
                        <xsd:sequence>
                              <xsd:sequence>
                                    <xsd:element name="firstName" type="xsd:string"/>
                                    <xsd:element name="familyName" type="xsd:string"/>
                                    <xsd:element name="age" type="tns:AgeType"/>
                                    <xsd:element ref="tns:detailsBrotherList"/>
                              </xsd:sequence>
                        </xsd:sequence>
                  </xsd:complexType>
            </xsd:schema>
      </types>
      <portType name="wsGetUserInfosPort">
            <operation name="getUserInfos">
                  <documentation>@param string $name</documentation>
                  <input message="tns:getUserInfosIn"/>
                  <output message="tns:getUserInfosOut"/>
            </operation>
      </portType>
      <binding name="wsGetUserInfosBinding" type="tns:wsGetUserInfosPort">
            <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
            <operation name="getUserInfos">
                  <soap:operation soapAction="http://www.zend.com/getuserinfo#getUserInfos"/>
                  <input>
                        <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.zend.com/getuserinfo"/>
                  </input>
                  <output>
                        <soap:body use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://www.zend.com/getuserinfo"/>
                  </output>
            </operation>
```

```
</binding>
<service name="wsGetUserInfosService">
      <port name="wsGetUserInfosPort" binding="tns:wsGetUserInfosBinding">
            <soap:address location="http://www.zend.com/getuserinfo"/>
      </port>
</service>
<message name="getUserInfosIn">
      <part name="name" type="xsd:string"/>
</message>
<message name="getUserInfosOut">
```

```
            <part name="return" type="tns:wsGetUserInfosResult"/>
      </message>
</definitions>
```

Generated part of XSD for Use Case 2

```
<xsd:simpleType name="FilterTypeString">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="test1"/>
        <xsd:enumeration value="test2"/>
        <xsd:enumeration value="test3"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="FilterTypeInteger">
    <xsd:restriction base="xsd:positiveInteger">
        <xsd:enumeration value="1"/>
        <xsd:enumeration value="2"/>
        <xsd:enumeration value="3"/>
    </xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="FilterType" id="10">
    <xsd:union id="11" memberTypes="tns:FilterTypeString tns:FilterTypeInteger"/>
</xsd:simpleType>
<xsd:simpleType name="FilterListType">
    <xsd:union itemType="tns:FilterType"/>
</xsd:simpleType>
```

9. Class Skeletons

```


class Zend_Soap_Wsdl_Strategy_DefaultComplexType extends Zend_Soap_Wsdl_Strategy_Abstract
{

    /**
     * @var array
     */
    protected $_simpleTypeList = array();

    /**
     * Add a complex type by recursivly using all the class properties fetched via Reflection.
     *
     * @param   string  $type Name of the class to be specified
     * @return  string  XSD Type for the given PHP type
     */
    public function addComplexType($type)
    {
        if(!class_exists($type)) {
            require_once "Zend/Soap/Wsdl/Exception.php";
            throw new Zend_Soap_Wsdl_Exception(sprintf(
                "Cannot add a complex type %s that is not an object or where ".
                "class could not be found in 'DefaultComplexType' strategy.", $type
            ));
        }

        $dom = $this->getContext()->toDomDocument();
        $class = new ReflectionClass($type);

        $defaultProperties = $class->getDefaultProperties();

        /*
         * If, in fact, we treat a simple type, we avoid rest of treatment
```

```
         * of complex type and focuse on add a simple type to the document.
         */
        if (preg_match_all('/@xsd\s+simpleType([ \t\f]+(.*))?/m', $class->getDocComment(), $matches))
{
            $this->_addSimpleType($type, $class, $matches[1][0]);
            return "tns:$type";
        }

        /*
         * In some case, should want to keep direct declaration as complexType
         * and not wrapping it inside of element because we don't need
         * to make reference on it and/or add restriction like minOccurs/maxOccurs.
         * It's the case at least for the Response element which need a type in it's
         * definition using class specified in return of the Doc in function added to
         * webservice.
         */
        $wantType = true;
        if (preg_match_all('/@xsd\s+element([ \t\f]+(.*))?/m', $class->getDocComment(), $matches)) {

            $wantType = false;
        } else if (preg_match_all('/@xsd\s+complexType([ \t\f]+(.*))?/m', $class->getDocComment(),
$matches)) {
        }

        $complexType = $dom->createElement('xsd:complexType');

        /*
         * Skipp element level creation if we don't want to
         * wrap the complexType in element.
         */
        if ($wantType) {
            $complexType->setAttribute('name', $type);
            $rootTypeElement = $complexType;
        } else {
            $rootTypeElement = $dom->createElement('xsd:element');
            $rootTypeElement->setAttribute('name', $type);
        }

        //$matches[1][0]=substr($matches[1][0], 0, -1);
        $this->_addAttributes($matches[1][0], $rootTypeElement);

        /*
         * Inside of complexType, choose the wished container.
         * Like sequence or choice or all, depending on restriction
         * you want on each sub-element. If nothing specified, keep all.
         */
        if (preg_match_all('/@xsd\s+([^\s]+)([ \t\f]+(.*))?/m', $class->getDocComment(), $matches)) {

            foreach($matches[1] as $key => $value) {
                switch ($matches[1][$key]) {
                    case 'element' :
                    case 'complexType' :
                        break;
                    default :
                        $all = $dom->createElement('xsd:' . $matches[1][$key]);
                        $this->_addAttributes($matches[2][$key], $all);
                        break;
                }
            }
            if(!isset($all)) {
                $all = $dom->createElement('xsd:all');
            }
        } else {
            $all = $dom->createElement('xsd:all');
        }

        /*
         * Keep copy of initial $all position in the tree for
         * easy backup tree point later.
         */
```

```
        $initAll = $all;

        /*
         * Keep instant copy of just before new $all position for
         * easy append of new element.
         */
        $oldAll = $all;

        /*
         * Start looking at each public property of a class.
         * Add ref in case of complexType exist.
         * Add element in case of base type.
         * Add wrapper at any point if we find sequence, choice or all.
         * Add restriction on each element if specified in array in Doc.
         */
        foreach ($class->getProperties() as $property) {
            if ($property->isPublic()
                 && preg_match_all('/@([^\s]+)\s+([^\s]+)([ \t\f]+(.*))?/m',
$property->getDocComment(), $matches)) {
                foreach ($matches[0] as $key => $someValue) {
                    switch ($matches[1][$key]) {
                        case 'var' :
                            $element = $dom->createElement('xsd:element');
                            if (class_exists(trim($matches[2][$key])) &&
$this->_isElement(trim($matches[2][$key]))
                                && !$this->_isSimpleType(trim($matches[2][$key]))) {
                                /*
                                 * Known issu: Do not use ref style for now, it can't work
                                 * with how is generated the soap response currently.
                                 * With reference, should add the namespace in tag name of the
                                 * element in the soapresponse. So it will NOT pass the validation
                                 * currently. You'll get an unresolved symbol.
                                 */
                                $element->setAttribute('ref',
$this->getContext()->getType(trim($matches[2][$key])));
                            } else {
                                $element->setAttribute('name', $property->getName());
                                $element->setAttribute('type',
$this->getContext()->getType(trim($matches[2][$key])));
                            }
                            $this->_addAttributes($matches[3][$key], $element);
                            // If the default value is null, then this property is nillable.
                            if ($defaultProperties[$property->getName()] === null) {
                                $element->setAttribute('nillable', 'true');
                            }
                            $all->appendChild($element);
                            break;
                        case 'xsd' :
                            if (preg_match_all('/@xsd\s+([^\s]+)\s+([^\s]+)([ \t\f]+(.*))?/m',
$matches[0][$key], $subMatches)) {
                                switch ($subMatches[1][0]) {
                                    default :
                                        if ($subMatches[2][0] == 'start') {
                                            $all = $dom->createElement('xsd:' . $subMatches[1][0]);

                                            $this->_addAttributes($subMatches[3][0], $all);
                                            $oldAll->appendChild($all);
                                            $oldAll = $all;
                                        } else if ($subMatches[2][0] == 'end') {
                                            $all = $all->parentNode;
                                            $oldAll = $oldAll->parentNode;
                                        }
                                        break;
                                }
                            }
                            break;
                    }
                }
            }
        }
```

```
        /*
         * Switch back to initial point
         */
        $all = $initAll;

        /*
         * Skipp append of first element wrapper if we just want a complexType definition
         */
        if ($wantType) {
            $complexType->appendChild($all);
        } else {
            $complexType->appendChild($all);
            $rootTypeElement->appendChild($complexType);
        }

        $this->getContext()->getSchema()->appendChild($rootTypeElement);
        $this->getContext()->addType($type);

        return "tns:$type";
    }

     /**
     * Add attributes to any element found via Reflection.
     *
     * @param   string  $attributes List of attributes to add to element
     * @param   DOMNode $element Element to add attributes on
     * @return  bool    true on success or false on failure
     */
    protected function _addAttributes($attributes, $element)
    {
        if ($attributes != '') {
            $attArgs = null;
            eval ('$attArgs = ' . $attributes . ';');
            if (is_array($attArgs)) {
                foreach ($attArgs as $attribut => $value) {
                    if ($attribut == 'base') {
                        $valueDetails=explode(':', $value);
                        if ($valueDetails[0] == 'tns' && class_exists($valueDetails[1])) {
                            $class = new ReflectionClass($valueDetails[1]);
                            if (preg_match_all('/@xsd\s+simpleType([ \t\f]+(.*))?/m',
$class->getDocComment(), $subMatches)) {
                                $this->_addSimpleType($valueDetails[1], $class, $subMatches[1][0]);
                                $element->setAttribute($attribut, $value);
                            }
                        } else if ($valueDetails[0] == 'xsd') {
                            $element->setAttribute($attribut, $value);
                        }
                    } else {
                        $element->setAttribute($attribut, $value);
                    }
                }
                return true;
            }
        }
        return false;
    }

    /**
     * Add simpleType
     *
     * @param   string  $name Name of simpleType
     * @param   object  $classReflection Result of reflection on class definition of simpleType
     * @param   string  $attributs Attributs to add to simple type tag
     * @return  bool    true on success or false on failure
     */
    protected function _addSimpleType($name, $classReflection, $attributs)
    {
        if(in_array($name, $this->_simpleTypeList))
            return true;
```

```
        $this->_simpleTypeList[] = $name;

        $dom = $this->getContext()->toDomDocument();

        $rootTypeElement = $dom->createElement('xsd:simpleType');
        $rootTypeElement->setAttribute('name', $name);

        if ($attributs != '')
            $this->_addAttributes($attributs, $rootTypeElement);

        if (preg_match_all('/@xsd\s+([^\s]+)([ \t\f]+(.*))?/m', $classReflection->getDocComment(),
$firstMatches)) {
            foreach ($firstMatches[0] as $firstKey => $someValue) {
                switch ($firstMatches[1][$firstKey]) {
                    case 'restriction' :
                        $all = $dom->createElement('xsd:restriction');
                        $this->_addAttributes($firstMatches[2][$firstKey], $all);

                        foreach ($classReflection->getProperties() as $property) {
                            if ($property->isPublic()
                                 && preg_match_all('/@xsd\s+([^\s]+)([ \t\f]+(.*))?/m',
$property->getDocComment(), $matches)) {
                                foreach ($matches[0] as $key => $someValue) {
                                    switch ($matches[1][$key]) {
                                        default :
                                            $enum = $dom->createElement('xsd:' . $matches[1][$key]);

                                            $this->_addAttributes($matches[2][$key], $enum);
                                            $all->appendChild($enum);
                                            break;
                                    }
                                }
                            }
                        }
                        break;
                    case 'union' :
                        $all = $dom->createElement('xsd:union');
                        $this->_addAttributes($firstMatches[2][$firstKey], $all);
                        foreach ($classReflection->getProperties() as $property) {
                            if ($property->isPublic()
                                 && preg_match_all('/@xsd\s+simpleType([ \t\f]+(.*))?/m',
$property->getDocComment(), $matches)) {
                                $list = array();
                                foreach ($matches[0] as $key => $someValue) {
                                    if (class_exists($matches[1][$key])) {
                                        $list[] = 'tns:' . $matches[1][$key];
                                        $class = new ReflectionClass($matches[1][$key]);
                                        if (preg_match_all('/@xsd\s+simpleType([ \t\f]+(.*))?/m',
$class->getDocComment(), $subMatches)) {
                                            $this->_addSimpleType($matches[1][$key], $class,
$subMatches[1][0]);
                                        }
                                    } else {
                                        $list[] = 'xsd:' . $matches[1][$key];
                                    }
                                }
                            }
                        }
                        if (count($list > 0))
                            $all->setAttribute('memberTypes', implode(" ", $list));
                        break;
                    case 'list' :
                        $all = $dom->createElement('xsd:list');
                        $this->_addAttributes($firstMatches[2][$firstKey], $all);
                        foreach ($classReflection->getProperties() as $property) {
                            if ($property->isPublic()
                                 && preg_match_all('/@xsd\s+itemType([ \t\f]+(.*))?/m',
$property->getDocComment(), $matches)) {
                                $list = array();
```

```
                                foreach ($matches[0] as $key => $someValue) {
                                    if (class_exists($matches[1][$key])) {
                                        $list[] = 'tns:' . $matches[1][$key];
                                        $class = new ReflectionClass($matches[1][$key]);
                                        if (preg_match_all('/@xsd\s+simpleType([ \t\f]+(.*))?/m',
$class->getDocComment(), $subMatches)) {
                                            $this->_addSimpleType($matches[1][$key], $class,
$subMatches[1][0]);
                                        }
                                    }
                                }
                            }
                        }
                        if (count($list > 0))
                            $all->setAttribute('itemType', $list[0]);
                        break;
                }
            }
        } else {
            return false;
        }

        $rootTypeElement->appendChild($all);
        $this->getContext()->getSchema()->appendChild($rootTypeElement);
        $this->getContext()->addType($name);
    }

    /**
     * Check if it is simpleType class
     *
     * @param   string  $name Name of simpleType
     * @return  bool    true on success or false on failure
     */
    protected function _isSimpleType($name)
    {
        $class = new ReflectionClass($name);

        if (preg_match_all('/@xsd[\s]+simpleType([ \t\f]+(.*))?/m', $class->getDocComment(),
$matches)) {
            return true;
        }
        return false;
    }

    /**
     * Check if it is element or complexType
     *
     * @param   string  $name Name of simpleType
     * @return  bool    true on success or false on failure
     */
    protected function _isElement($name)
    {
        $class = new ReflectionClass($name);

        if (preg_match_all('/@xsd[\s]+element([ \t\f]+(.*))?/m', $class->getDocComment(), $matches)) {

            return true;
        }
```

```
            return false;
        }
    }
```

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