exolab.org

CASTOR XML

SOURCE CODE GENERATOR

USER DOCUMENT

Author: Arnaud Blandin [blandin@intalio.com]

Contributions:

Keith Visco [kvisco@intalio.com]

Revision: July 11, 2001

1 Abstract

As a complete XML[1] data binding framework, Castor XML offers the ability to generate Java code from an XML Schema.

The aim of this document is to describe the Castor XML Source Code Generator, including the main options and features, as well as to illustrate the mechanism used to generate java classes with a simple example. Open issues of the project are also discussed.

1: all bold names (except in section 2.3) are defined in the glossary.

1 Why a Source Code Generator?

1.1 XML Data Binding

Many current applications which manipulates XML documents rely on XML Schemas which define the structure, the content and even the meaning of these XML documents.

In order to deal with the XML 'constraints' defined in the schema, applications need some tools to create and manipulate XML documents that are instances of the given XML Schema.

Such tools might be written using the SAX API or the DOM API, however these approaches are more focused on the structure of an XML document than the data itself.

Moreover all data in these APIs are treated as strings and must likely need to be cast to an appropriate data type.

It would be much easier if these applications could map directly an XML document to its in-memory object representation that contains all the information provided by the XML Schema, this is what we call XML data binding.

1.2The role of the source generator

To represent the data model of an XML document in memory, developers need to hard-code the description of the XML document. They need to describe the structure and the data of the document provided by the XML Schema.

Sometimes it could be easy when you only need to map a String or a Boolean, you can find the exact mapping in any Object Oriented language but when it is time to describe a more complex structure with some inner XML Schema types it can become very tedious and complex.

The aim of the source generator is to provide the necessary code to describe XML instances of a specific XML Schema with the proper fields and access methods.

To sum up, we can draw a parallel between the relations XML Schema-XML and Class-Object: an XML document is an ***instance*** of an XML Schema and an Object is an ***instance*** of a Class. Thus to represent an XML document as an Object in memory, we need to provide the Class that describes this object.

The Source Code Generator is merely generating the code for this class.

The Castor XML Source Code Generator – from now on referred to the Source Generator –generates Java source code from a W3C XML Schema. The generated source includes an object model of the schema as well as the necessary Class Descriptors used by the marshalling framework to obtain information about the generated classes.

2 Usage, options & XML Schema support

2.1 Usage

The source generator can be used as a command-line tool that can simply be invoked by calling :

```
java org.exolab.castor.builder.SourceGenerator –options
```

or using the script files SourceGenerator.bat or SourceGenerator.sh.

The API can also be used, for more information refer to the Javadoc of the SourceGenerator class.

Note: the generated source files need to be compiled by hand.

2.2 Source Generator Options

The source code generator has a number of different options which may be set. Some of these are done using the command line, and others are done using a properties file (`castorbuilder.properties`).

2.2.1 Command Line Options

Option	Args	Description	Status
i	filename	The input XML Schema file	Required
package	***package-name***	The package for the generated source	Optional
dest	***path***	The destination in which to put the generated source	Optional
line-separator	***unix | mac | win***	Sets the line separator style for the desired platform. This is useful if you are generating source on one platform, but will be compiling/modifying on another platform.	Optional
types	***type-factory***	Sets which type factory to use. This is useful if you want JDK 1.2 collections instead of JDK 1.1) (see below)	Optional
h		Shows the help/usage screen	Optional
f		Forces the source generator to supress all non-fatal errors, such as overwriting of pre-existing files.	Optional
nodesc		Do not generate the class descriptors	Optional
nomarshall		Do not generate the marshalling framework methods (marshall, unmarshall, validate).	Optional
testable		Generate specific methods used by the Castor Testing Framework.	Optional

2.2.2 Collection types:

The source code generator has the ability to use the following types of collections when generating source code:

- Java 1.1 (default) java.util.Vector.
- Java 1.2: if the option is types –j2, collection type will be java.util.Collections implemented as ArrayList.
- ODMG 3.0: if the option is types –odmg, collection type will be odmg.Darray.

2.2.3 Advanced options

These options are set up in the `org/exolab/castor/builder/castorbuilder.properties` file.

2.2.3.1 Bound Properties

Since version: 0.8.9

Bound properties are "properties" of a class, which when updated will send out a java.beans.PropertyChangeEvent to all registered java.beans.PropertyChangeListeners.

To enable bound properties, uncomment the appropriate line in the "org/exolab/castor/builder/castorbuilder.properties" file:

```
        # To enable bound properties uncomment the
        # following  line.
        # Please note that currently *all* fields will
        # be treated as bound properties
        # when enabled. This will change in the future
        # when we introduce fine grained control over
        # each class and its properties.
        #
        #org.exolab.castor.builder.boundproperties=true
```

When enabled, all properties will be treated as bound properties. For each class that is generated a setPropertyChangeListener method is created as follows:

```
/**
 * Registers a PropertyChangeListener with this class.
 * @param pcl The PropertyChangeListener to register.
 */

public void addPropertyChangeListener(java.beans.PropertyChangeListener pcl)
{
     propertyChangeListeners.addElement(pcl);
} //-- void addPropertyChangeListener(java.beans.PropertyChangeListener)
```

Whenever a property of the class is changed, a PropertyChangeEvent will be sent to all registered listeners. The property name, the old value, and the new value will be set in the PropertyChangeEvent.

Note: To prevent unnecessary overhead, if the property is a collection, the old value will be null.

2.2.3.2 Class Creation/Mapping

Since version: 0.8.9

The source generator can treat the XML Schema structures such as complexType and element in two main ways.

The first, and currently default method is called the "element" method. The other is called the "type" method.

In the following we will illustrate the class creation with the following schema:

```
<?xml version='1.0'?>
<schema xmlns="http://www.w3.org/2000/10/XMLSchema"
        targetNamespace="http://castor.exolab.org/Examples">

   <complexType name="A">
        <sequence>
            <element name="B" type="string"/>
            <element name="C" type="string"/>
        </sequence>
    </complexType>
</schema>
```

The 'element' method

The "element" method creates classes for all elements whose type is a complexType. Abstract classes are created for all top-level complexTypes. Any elements whose type is a top-level type will have a new class created that extends the abstract class which was generated for that top-level complexType.

Classes are not created for elements whose type is a simpleType.

This approach tends to describe the structure of an XML Document that is a particular instance of the XML Schema used to generate the source code.

In this case the generated class will be:

```
/*
 * This class was automatically generated with
 * <a href="http://castor.exolab.org">Castor 0.8.10</a>, using an
 * XML Schema.
 * $Id$
 */

package illustrate;

  //---------------------------------/
 //- Imported classes and packages -/
//---------------------------------/

import java.io.Reader;
import java.io.Serializable;
import java.io.Writer;
import org.exolab.castor.xml.*;
import org.exolab.castor.xml.ValidationException;

/**
 *
 * @version $Revision$ $Date$
**/
public abstract class A implements java.io.Serializable {

      //--------------------------/
     //- Class/Member Variables -/
    //--------------------------/
    private java.lang.String _b;
    private java.lang.String _c;
```

The 'type' method

The "type" method creates classes for all top-level complexTypes, or elements that contain an "anonymous" (in-lined) complexType.

Classes will not be generated for elements whose type is a top-level type.

This second approach tries to produce a class hierarchy that represents the XML Schema itself more than an instance of this XML Schema.

In this case the generated class will be:

```
package illustrate;


  //---------------------------------/

 //- Imported classes and packages -/

//---------------------------------/


import java.io.Reader;

import java.io.Serializable;

import java.io.Writer;

import org.exolab.castor.xml.*;

import org.exolab.castor.xml.MarshalException;

import org.exolab.castor.xml.ValidationException;

import org.xml.sax.DocumentHandler;


/**

 *

 * @version $Revision$ $Date$

**/

public class A implements java.io.Serializable {


      //--------------------------/

     //- Class/Member Variables -/

    //--------------------------/

    private java.lang.String _b;
    private java.lang.String _c;
```

To change the "method" of class creation simply edit the `castorbuilder.properties` file:

```
# Java class mapping of <xsd:element>'s and <xsd:complexType>'s
#
#org.exolab.castor.builder.javaclassmapping=element
```

2.2.3.3 Setting a super class

Since version: 0.8.10

The source generator enables the user to set a super class to all the generated classes (of course class descriptors are not concerned by this option).

To set the super class, edit the `castorbuilder.properties` file:

```
# This property allows one to specify the super class of *all*
# generated classes
#
#org.exolab.castor.builder.superclass=com.xyz.BaseObject
```

2.2.3.4 Map XML Namespaces to Java packages

Since version: 0.9.0

An XML Schema grammar is identified by a namespace. For data-binding purpose it may be convenient to map namespaces to Java packages.

To allow the mapping between namespaces and Java packages , edit the castorbuilder.properties file :

```
 # XML namespace mapping to Java packages
 #
 #org.exolab.castor.builder.nspackages=\
  http://www.xyz.com/schemas/project=com.xyz.schemas.project,\
  http://www.xyz.com/schemas/person=com.xyz.schemas.person
```

2.2.3.5 Generate equals() method

Since version: 0.9.1

The Source Generator can override the 'equals' method for the generated objects.

Note: the hashcode() method is currently not overriden.

To generate the equals() method, edit the castorbuilder.properties file:

```
# Set to true if you want to generate the equals method

# for each generated class

# false by default

#org.exolab.castor.builder.equalsmethod=true
```

2.2.3.6 Maps primitive types to wrapper objects

Since version: 0.9.4

It may be convenient to use java objects instead of primitives, the Source Generator provides a way to do it. Thus the following mapping can be used:

- boolean to java.lang.Boolean
- byte to java.lang.Byte
- double to java.lang.Double
- float to java.lang.Float
- int and integer to java.lang.Integer
- long to java.lang.Long
- short to java.lang.Short

To enable this property, edit the castor builder.properties file:

```
# Set to true if you want to use Object Wrappers instead

# of primitives (e.g Float instead of float).

# false by default.

#org.exolab.castor.builder.primitivetowrapper=false
```

2.3 XML Schema support

The source generator supports the W3C XML Schema Recommendation document (05/02/2001).

Roughly speaking the source generator maps a XML Schema type to a corresponding Java type.

It happens that a Schema type does not have its corresponding one in Java. Thus the Source Generator uses Castor implementation of these specific types (located in the package `org.exolab.castor.types`). For instance the Duration type is implemented directly in Castor.

Many built-in types are supported but not all of them. You will find below a detailed list of supported built-in types followed by comments about the support.

Remember that the representation of XML Schema datatypes does not try to fit exactly the W3C XML Schema specifications, the aim is to map an XML Schema type to the java type that fit the most to the XML Schema type.

For instance this can explain why you won’t find the support for the ‘pattern’ facet for the date/time support.

2.3.1 Built-in types

2.3.1.1 Primitive Datatypes

The bold names refer to features supported by the Source Generator. The ***italic*** names refer to the ***unsupported*** dataypes.

XML Schema Datatypes	Facets	Java corresponding types
string	length minLength maxLength pattern enumeration **whiteSpace**	java.lang.String
boolean	pattern whiteSpace	primitive boolean type by default (see 2.2.3.6)
decimal	totalDigits fractionDigits pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	java.math.BigDecimal
float	pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	primitive float type by default (see 2.2.3.6)
double	pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	primitive double type by default (see 2.2.3.6)

XML Schema Datatypes	Facets	Java corresponding types
duration	pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	org.exolab.castor.types.Duration
dateTime	pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	java.util.Date
time	pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	org.exolab.castor.types.Time
date	pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	org.exolab.castor.types.Date
gYearMonth	pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	org.exolab.castor.types.GYearMonth
gYear	pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	org.exolab.castor.types.GYear
gMonthDay	pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	org.exolab.castor.types.GMonthDay

XML Schema Datatypes	Facets	Java corresponding types
gDay	pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	org.exolab.castor.types.GDay
gMonth	pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	org.exolab.castor.types.GMonth
hexBinary	length max/min length pattern enumeration whiteSpace	primitive byte array
base64Binary	length max/min length pattern enumeration whiteSpace	primitive byte array
anyURI	length max/min length pattern whiteSpace enumeration	java.lang.String
Qname	length max/min length pattern whiteSpace enumeration max/min Exclusive max/min Inclusive	Java.lang.String
NOTATION	length max/min length pattern whiteSpace enumeration	

2.3.1.2 Derived datatypes

The bold names refer to features supported by the Source Generator. The ***italic*** names refer to the ***unsupported*** dataypes

XML Schema Datatypes	Facets	Java corresponding types
normalizedString	length max/min Length pattern enumeration whiteSpace	Java.lang.String
token	length max/min Length pattern enumeration whiteSpace	
language	length max/min Length pattern enumeration whiteSpace	
NMTOKEN	length max/min Length pattern enumeration whiteSpace	java.lang.String
Name	length max/min Length pattern enumeration whiteSpace	

XML Schema Datatypes	Facets	Java corresponding types
NCName	length max/min Length pattern enumeration whiteSpace	java.lang.String
ID	length max/min Length pattern enumeration whiteSpace	java.lang.String
IDREF	length max/min Length pattern enumeration whiteSpace	java.lang.Object
IDREFS	length max/min Length enumeration whiteSpace	java.util.Vector of IDREF
ENTITY	length max/min Length pattern enumeration whiteSpace	
ENTITIES	length max/min Length enumeration whiteSpace	

XML Schema Datatypes	Facets	Java corresponding types
integer	totalDigits fractionDigits pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	primitive int type by default (see 2.2.3.6)
nonPositiveInteger	totalDigits fractionDigits pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	primitive int type by default (see 2.2.3.6)
negativeInteger	totalDigits fractionDigits pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	primitive int type by default (see 2.2.3.6)
long	totalDigits fractionDigits pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	primitive long type by default (see 2.2.3.6)

XML Schema Datatypes	Facets	Java corresponding types
int	totalDigits fractionDigits pattern enumeration whiteSpace max/min Exclusive max/min Inclusive max/min Inclusive	primitive int type by default (see 2.2.3.6)
short	totalDigits fractionDigits pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	primitive short type by default (see 2.2.3.6)
byte	totalDigits fractionDigits pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	primitive byte type by default (see 2.2.3.6)

XML Schema Datatypes	Facets	Java corresponding types
nonNegativeInteger	totalDigits fractionDigits pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	primitive int type by default (see 2.2.3.6)
unsignedLong	totalDigits fractionDigits pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	
unsignedInt	totalDigits fractionDigits pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	

XML Schema Datatypes	Facets	Java corresponding types
unsignedShort	totalDigits fractionDigits pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	
unsignedByte	totalDigits fractionDigits pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	
positiveInteger	totalDigits fractionDigits pattern enumeration whiteSpace max/min Exclusive max/min Inclusive	primitive int type by default (see 2.2.3.6)

2.3.1.2 Conclusion – Comments

The Source Generator can handle 33 of the 43 XML Schema Datatypes with however some restrictions.

Primitive datatypes

The Source Generator supports 18 of the 19 W3C XML Schema primitive datatypes.

However this support is not complete and sometimes full support is not required.

- duration[section 3.2.6 XML Schema Part 2 :datatypes, W3C Recommendation]

The order relation differs from the W3C partial relation order. The relation provided by Castor is a total relation and provides the same results expected with the W3C order relation.

- hexBinary [section 3.2.15 XML Schema Part 2 :datatypes, W3C Recommendation]

A binary type is by default treated as with a base64 encoding.

- anyURI [section 3.2.9 XML Schema Part 2 :datatypes, W3C Recommendation]

A uriReference is currently represented as a string. No specific validation is done.

- Qname [section 3.2.18 XML Schema Part 2 :datatypes, W3C Recommendation]

This type is only represented as a String. A full representation of it requires a full representation of anyURI

- NOTATION [section 3.2.19 XML Schema Part 2 :datatypes, W3C CR]

  Not Supported

This type is provided by XML Schema to allow backward compatibility with DTD. DTD tends to be replaced by XML Schemas and the use of this type is not widespread enough for requiring support in the Source Generator.

Note: The date/time datatypes are supported when used as

elements as weel as attributes.

Derived datatypes

The Source Generator is supporting 15 of the 24 XML Schema derived datatypes

- token [section 3.3.2 XML Schema Part 2:datatypes, W3C Recommendation]

Not Supported

The use of token is not widespread, its derived type (NMToken, Name) are more important.

- language [section 3.3.3 XML Schema Part 2:datatypes, W3C Recommendation]

Not Supported

- Name [section 3.3.6 XML Schema Part 2:datatypes, W3C Recommendation]

Not Supported

- ENTITY [section 3.3.11 XML Schema Part 2:datatypes, W3C Recommendation]

Not Supported

c.f. comments on NOTATION.

- ENTITIES [section 3.3.12 XML Schema Part 2:datatypes, W3C Recommendation]

Not Supported

c.f. comments on NOTATION.

- unsignedLong, unsignedInt, unsignedShort, unsignedByte [section 3.3.19, 3.3.20,3.3.21, 3.3.22 XML Schema Part 2: datatypes, W3C CR]

Not Supported

In Java, all numeric types are signed, it is not possible to deal properly with unsigned numeric types.

2.3.2 Structure

The following section is based on the XML Schema Part 1: structure, W3C Recommendation.

Each paragraph number refers to the W3C document.

2.3.2.1 Schema Component: Element Declaration

- § 3.3.1 Element Declaration details

  Supported:

  - {name}
  - {target namespace}
  - {type definition}
  - {scope}
  - {annotation}
  - {value constraint}

  Unsupported:

  - {abstract}
  - {nillable}
  - {identity-constraint definitions}
  - {substitution group affiliation}
  - {substitution group exclusions}
  - {disallowed substitutions}

- § 3.3.2 XML Representation of Element Declaration Schema Components

  Unsupported features appear in *italics* :

<element
  abstract = boolean : false
  block = (#all | List of (substitution | extension | restriction))
  default = string
  final = (#all | List of (extension | restriction))
  fixed = string
  form = (qualified | unqualified) : unqualified
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  name = NCName
  nillable = boolean : false
  ref = QName
  substitutionGroup = QName
  type = QName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation?, ((simpleType | complexType)?, (*key* | *keyref* | *unique*)*))
</element>

2.3.2.2 Schema Component: Complex Type Definition

- § 3.4.1 Complex Type Definition Details

  Supported:

  - {name}
  - {target namespace}
  - {base type definition}
  - {derivation method}
  - {abstract}
  - {attribute use pairs}
  - {content type}
  - {annotations}

  Unsupported:

  - {final}
  - {attribute wildcard}
  - {prohibited substitutions}

- § 3.4.2 XML Representation of Complex Type Definition Schema Components.

  Unsupported features appear in ***italics:***

```
<complexType
 abstract=boolean : false
 block = (#all | List of (extension | restriction))
 final = (#all | List of (extension | restriction))
 id = ID
 mixed = boolean : false
 name = NCName
 {any attributes with non-schema namespace . . .}>
 Content: (annotation? , (simpleContent | complexContent |
((group | all|choice | sequence)? , ((attribute | attributeGroup)* , anyAttribute?))))
</complexType>
```

2.3.2.3 Schema Component: Attribute Declaration

- § 3.5.1 Attribute Declaration details

  Supported:

  - {name}
  - {target namespace}
  - {type definition}
  - {scope}
  - {value constraint}
  - {annotation}

- § 3.5.2 XML Representations of Attribute Declaration Schema Components

  Unsupported features appear in *italics*.

```
<attribute
      default = string
      fixed = string
      form = (qualified | unqualified)
      id = ID
      name = NCName
      ref = QName
      type = QName
      use = (prohibitied | optional | required ): optional
      value = string
      {any attributes with non-schema namespace . . . }>
      Content: (annotation?, (simpleType?))
</attribute>
```

2.3.2.4 Schema Component: Attribute Group Definition

- § 3.6.1 Attribute Group Definition Details

  Supported:

- {name}
- {target namespace}
- {attribute use pairs}
- {annotation}

Unsupported:

- {attribute wildcard}

- § 3.6.2 XML Representation of Attribute Group Definition Schema Components.

  Unsupported features appear in *italics*:

```
<attributeGroup
 id = ID
 name = NCName
 ref = QName
 {any attributes with non-schema namespace . . .}>
 Content: (annotation? , ((attribute | attributeGroup)* , anyAttribute?))
</attributeGroup>
```

2.3.2.5 Schema Component: Model Group Definition

- § 3.7.1 Model Group Definition Details

  Supported:

  - {name}
  - {target namespace}
  - {model group}
  - {annotation}

- § 3.7.2 XML Representation of Model Group Definition Schema Components.

  Unsupported features appear in *italics*:

```
<group
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded) : 1
  minOccurs = nonNegativeInteger : 1
  name = NCName
  ref = QName
  {any attributes with non-schema namespace . . .}>
  Content: (annotation? , (all | choice | sequence)?)
</group>
```

2.3.2.6 Schema Component: Model Group

- § 3.8.1 Model Group Details

  Supported:

  - {compositor}

  - {particles}

  - {annotation}

- § 3.8.2 XML Representation of Model Group Schema Components.

  Unsupported features appear in *italics*:

```
<all
  id = ID
  maxOccurs = 1 : 1
  minOccurs = 1 : 1
  {any attributes with non-schema namespace . . .}>
  Content: (annotation? , element*)
</all>
```

```
<choice
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded)  : 1
  minOccurs = nonNegativeInteger : 1
  {any attributes with non-schema namespace . . .}>
  Content: (annotation? , (element | group | choice
| sequence | any)*)
</choice>
```

```
<sequence
  id = ID
  maxOccurs = (nonNegativeInteger | unbounded)  : 1
  minOccurs = nonNegativeInteger : 1
  {any attributes with non-schema namespace . . .}>
  Content: (annotation? , (element | group | choice
| sequence | any)*)
</sequence>
```

2.3.2.7 Schema Component: Particle

- § 3.9.1 Particle Details

  Supported:

  - {min occurs}

  - {max occurs}

  - {term}

2.3.2.8 Schema Component: Wildcard

Wildcards are currently not supported.

- § 3.10.1 Wildcard Details

  Unsupported:

  - {namespace constraint}

  - {process contents}

  - {annotation}

- § 3.10.2 XML representation of Wildcard schema component

```
<any
 id = ID
 maxOccurs = (nonNegativeInteger | unbounded) : 1
 minOccurs = nonNegativeInteger : 1
 namespace = ((##any | ##other) | List of (anyURI | (##targetNamespace | ##local)) ) : ##any
 processContents = (lax | skip | strict) : strict
 {any attributes with non-schema namespace . . .}>
 Content: (annotation?)
</any>
```

For more information on the support of Wildcard, you should refer to the documentation on AnyNode.

2.3.2.9 Schema Component: Identity-constraint Definition

Identity-constraints are currently not supported.

2.3.2.10 Schema Component: Notation Declaration

Notations are currently not supported.

2.3.2.11 Schema Component: Annotation

- § 3.13.1 Annotation Details

  Supported:

  - {application information}

  - {user information}

2.3.2.12 Schema Component: Simple Type Definition

- § 3.14.1 Simple Type Definition Details

  Supported:

  - {name}
  - {target namespace}
  - {base type definition}
  - {facets}
  - {variety}
  - {atomic}
  - {annotation}

  Unsupported:

  - {variety}
  - {list}
  - {union}

- **§ 3.14.2 (non-normative) XML Representation of Simple Type Definition Schema Components.**

  Unsupported features appear in ***italics***:

```
<simpleType
 final = (#all| (list | union| restriction) )
 id = ID
 name = NCName
 {any attributes with non-schema namespace . . .}>
 Content: (annotation? , ((list | restriction | union)))
</simpleType>
```

2.3.2.13 Schemas a Whole

- § 3.15.1Schema details

  Supported:

  - {type definitions}
  - {attribute declarations}
  - {attribute group definitions}
  - {model group definitions]
  - {annotations}

  Unsupported:

  - {notation declarations}

- § 3.15.2 XML Representations of Schemas

  Unsupported features appear in ***italics***

```
<schema
  attributeFormDefault = (qualified | unqualified) : unqualified
  blockDefault = ( #all | List of (substitution | extension | restriction))
  elementFormDefault = (qualified | unqualified) : unqualified
   id = ID
  targetNamespace = uriReference
  version = string
  {any attributes with non-schema namespace . . . }>
  Content: ((include | import | redefine | annotation)*,  ((attribute|attributeGroup | complexTy
element | group | notation | simpleType), annotation*)*)
</schema>
```

2.3.2.14 Conclusion – Comments

Castor can support – both in the Source Code Generator and the Schema Object Model– all the basic features of W3C XML Schema as defined in the Recommendation document.

Schema Object Model and Source Code Generator

Even if the Source Code Generator relies heavily on the Schema Object Model, their XML Schema support may differ.

In Element Declaration Component (see 2.3.2.1) the attribute `nillable` is supported only by the Schema Object Model.

The Wilcard *`anyAttribute`* (see 2.3.2.2 and 2.3.2.4) is supported only by the Schema Object Model.

Grouping support

Grouping support covers both Model Group Definitions (`<group>`) and Model Groups (`<all>, <choice>, <sequence>`).

In this section, we will call 'nested group', a Model Group whose first parent is another Model Group.

- For each top-level Model Group Definition, a class is generated either when using the 'element' mapping property or the 'type' one.(see section 2.2.3.2).
- If a group – nested or not – appears to have `maxOccurs` > 1 then a class is generated to represent the items contained in the group.
- For each nested group, a class is generated. The name of this generated class will follow the naming convention defined below:

```
ClassName ::= Name,Compositor+,Counter?
```

- 'Name' is the name of the top-level component in which the group is nested (element, complexType or group).
- 'Compositor' is the compositor of the nested group. For instance if a 'choice' is nested inside a 'sequence', the value of 'Compositor' will be 'SequenceChoice' ('Sequence'+'Choice').

Note: if the 'choice' is inside a Model Group and that Model Group parent is a Model Group Definition or a complexType then the value of'Compositor' will be only 'Choice'.

- 'Counter' is simply a counter that prevents from naming collision.

For example, the following XML Schema part:

```
<xsd:complexType name='InvoiceType'>
    <xsd:sequence>
        <xsd:group ref="Person"/>
        <xsd:choice maxOccurs='unbounded'>
            <xsd:element name="Item1" type="Item1Type"/>
            <xsd:element name="Item2" type="Item2Type"/>
            <xsd:element name="Item3" type="Item3Type"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:complexType>

<xsd:group name='Person'>
    <xsd:sequence>
        <xsd:element name="FirstName" type="xsd:string"/>
        <xsd:element name="LastName" type="xsd:string"/>
        <xsd:choice>
            <xsd:element name="HomeAddress" type="xsd:string"/>
            <xsd:element name="OfficeAddress" type="xsd:string"/>
        </xsd:choice>
    </xsd:sequence>
</xsd:group>
```

is composed of

- A top-level complexType that contains a nested group that can repeat itself (i.e. with maxOccurs='unbounded').
- A top-level group with a nested group.

According to the previous explanations, the generated classes will be (either with the 'type' or 'element' method, see section 2.2.3.2):

```
InvoiceType for the top-level complexType.

InvoiceTypeChoice for the nested ‘choice’ inside the ComplexType.

InvoiceTypeChoiceItem for the items inside the nested ‘choice’.

Person for the top-level group.

PersonChoice for the nested choice.
```

Inside the class ‘InvoiceTypeChoiceItem’, you’ll find a reference to Item1, Item2 and Item3.

2.4 Requirements

Castor XML is compliant with all JDK1.1 and above.

In order to run the Source Generator you’ll need to set the following libraries in your classpath.

2.4.1 Castor packages

- `org.exolab.castor.builder`
- `org.exolab.castor.builder.types`
- `org.exolab.castor.builder.util`
- `org.exolab.castor.mapping`
- `org.exolab.castor.types`
- `org.exolab.castor.xml`

2.4.2 External libraries

- `org.exolab.javasource` : this package contains a full description of a java object
- Xerces-Java parser version 1.4.0 and after: the Apache XML parser is the default parser used by Castor. Castor also relies on the Apache serializer shipped with Xerces. The version 1.4.0 is required.
- Jakarta regular expression.

These external libraries are included in the Castor distribution.

3 Example

In this section we illustrate the use of the Source Generator by explaining the generated classes from a given schema.

The Source Generator is going to be used with the "java class mapping" property (cf section 2.2.3.2) set to 'element' (default value).

3.1 The schema file

The input file is the schema file given with the Source Generator example in the distribution of Castor (under `/src/examples/SourceGenerator/invoice.xsd`)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                     targetNamespace="http://castor.exolab.org/Test/Invoice">


   <xsd:annotation>
      <xsd:documentation>
         This is a test XML Schema for Castor XML.
      </xsd:documentation>
   </xsd:annotation>


   <!--
       A simple representation of an invoice. This is simply an example
       and not meant to be an exact or even complete representation of an invoice.
     -->
   <xsd:element name="invoice">
      <xsd:annotation>
         <xsd:documentation>
             A simple representation of an invoice
         </xsd:documentation>
      </xsd:annotation>


      <xsd:complexType>
                 <xsd:sequence>
                     <xsd:element name="ship-to">
                        <xsd:complexType>
                            <xsd:group ref="customer"/>
                        </xsd:complexType>
                        </xsd:element>
                        <xsd:element ref="item" maxOccurs="unbounded" minOccurs="1"/>
                        <xsd:element ref="shipping-method"/>
                         <xsd:element ref="shipping-date"/>
```

```
            </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<!-- Description of a customer -->
<xsd:group name="customer">
     <xsd:sequence>
          <xsd:element name="name" type="xsd:string"/>
          <xsd:element ref="address"/>
          <xsd:element name="phone" type="TelephoneNumberType"/>
      </xsd:sequence>
 </xsd:group>

<!-- Description of an item -->
<xsd:element name="item">
    <xsd:complexType>
        <xsd:sequence>
             <xsd:element name="Quantity" type="xsd:integer" minOccurs="1" maxOccurs="1"/>
              <xsd:element name="Price" type="PriceType" minOccurs="1" maxOccurs="1"/>
         </xsd:sequence>
         <xsd:attributeGroup ref="ItemAttributes"/>
      </xsd:complexType>
  </xsd:element>

<!-- Shipping Method -->
<xsd:element name="shipping-method">
   <xsd:complexType>
       <xsd:sequence>
             <xsd:element name="carrier" type="xsd:string"/>
             <xsd:element name="option"  type="xsd:string"/>
             <xsd:element name="estimated-delivery" type="xsd:duration"/>
       </xsd:sequence>
   </xsd:complexType>
```

```
</xsd:element>


<!-- Shipping date -->
<xsd:element name="shipping-date">
    <xsd:complexType>
             <xsd:sequence>
                   <xsd:element name="date" type="xsd:date"/>
                   <xsd:element name="time" type="xsd:time"/>
             </xsd:sequence>
     </xsd:complexType>
</xsd:element>


<!-- A simple U.S. based Address structure -->
<xsd:element name="address">
   <xsd:annotation>
      <xsd:documentation>
         Represents a U.S. Address
      </xsd:documentation>
   </xsd:annotation>

   <xsd:complexType>
        <xsd:sequence>
            <!-- street address 1 -->
             <xsd:element name="street1" type="xsd:string"/>
             <!-- optional street address 2 -->
             <xsd:element name="street2" type="xsd:string" minOccurs="0"/>
             <!-- city-->
             <xsd:element name="city" type="xsd:string"/>
             <!-- state code -->
             <xsd:element name="state" type="stateCodeType"/>
             <!-- zip-code -->
             <xsd:element ref="zip-code"/>
         </xsd:sequence>
   </xsd:complexType>
</xsd:element>

<!-- A U.S. Zip Code -->
<xsd:element name="zip-code">
   <xsd:simpleType>
      <xsd:restriction base="xsd:string">
          <xsd:pattern value="[0-9]{5}(-[0-9]{4})?"/>
       </xsd:restriction>
   </xsd:simpleType>
</xsd:element>

<!-- State Code
     obviously not a valid state code....but this is just
     an example and I don't feel like creating all the valid
     ones.
 -->
<xsd:simpleType name="stateCodeType">
      <xsd:restriction base="xsd:string">
             <xsd:pattern value="[A-Z]{2}"/>
      </xsd:restriction>
</xsd:simpleType>

<!-- Telephone Number -->
<xsd:simpleType name="TelephoneNumberType">
      <xsd:restriction base="xsd:string">
           <xsd:length value="12"/>
           <xsd:pattern value="[0-9]{3}-[0-9]{3}-[0-9]{4}"/>
       </xsd:restriction>
</xsd:simpleType>

<!-- Cool price type -->
<xsd:simpleType name="PriceType">
```

```
        <xsd:restriction base="xsd:decimal">
                <xsd:fractionDigits value="2"/>
                <xsd:totalDigits value="5"/>
                <xsd:minInclusive value="1"/>
                <xsd:maxInclusive value="100"/>
        </xsd:restriction>
   </xsd:simpleType>

   <!-- The attributes for an Item -->
   <xsd:attributeGroup name="ItemAttributes">
        <xsd:attribute name="Id" type="xsd:ID" minOccurs="1" maxOccurs="1"/>
        <xsd:attribute name="InStock" type="xsd:boolean" default="false"/>
        <xsd:attribute name="Category" type="xsd:string" use="required"/>
    </xsd:attributeGroup>
</xsd:schema>
```

The structure of this schema is simple: it is composed of a top-level element which is a complexType with references to other elements inside.

This schema represents a simple invoice: an invoice is a ***customer*** (‘customer’ top-level group), an ***article*** (‘item’ element), a ***shipping method*** (‘shipping-method’ element) and a ***shipping date*** (‘shipping-date’ element).

Notice that the ‘ship-to’ element uses a reference to an ‘address’ element. This ‘address’ element is a top-level element that contains a reference to a non-top-level element (the ‘zip-code’ element).

At the end of the schema we have two simpleTypes for representing a telephone number and a price.

The Source Generator is used with the ‘element’ property set for class creation (c.f.section 2.2.3.2) so a class is going to be generated for all top-level elements. No classes are going to be generated for complexTypes and simpleTypes since the simpleType is not an enumeration.

To summarize, we can expect 7 classes : Invoice, Customer, Address, Item, ShipTo, ShippingMethod and ShippingDate and the 7 corresponding class descriptors. Note that a class is generated for the top-level group ‘customer’

To run the source generator and create the source from the invoice.xsd file in a package test, we just call in the command line:

```
java -cp %CP% org.exolab.castor.builder.SourceGenerator -i invoice.xsd  -
package test
```

3.2 The generated code

To simplify this example we now focus on the item element.

```
<!-- Description of an item -->
   <xsd:element name="item">
       <xsd:complexType>
           <xsd:sequence>
              <xsd:element name="Quantity" type="xsd:integer"minOccurs="1" maxOccurs="1"/>
              <xsd:element name="Price" type="PriceType" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
           <xsd:attributeGroup ref="ItemAttributes"/>
         </xsd:complexType>
   </xsd:element>

   <!-- Cool price type -->
   <xsd:simpleType name="PriceType">
       <xsd:restriction base="xsd:decimal">
           <xsd:fractionDigits value="2"/>
              <xsd:totalDigits value="5"/>
              <xsd:minInclusive value="1"/>
              <xsd:maxInclusive value="100"/>
        </xsd:restriction>
   </xsd:simpleType>

   <!-- The attributes for an Item -->
   <xsd:attributeGroup name="ItemAttributes">
        <xsd:attribute name="Id" type="xsd:ID" minOccurs="1" maxOccurs="1"/>
        <xsd:attribute name="InStock" type="xsd:boolean" default="false"/>
        <xsd:attribute name="Category" type="xsd:string" use="required"/>
    </xsd:attributeGroup>
```

To represent an Item object, we need to know its 'Id', the 'Quantity' ordered and the 'Price' for one item.

So we can expect to find a least three private variables: a `string` for the 'Id' element, an `int` for the 'quantity' element (see the section on XML Schema support if you want to see the mapping between a W3C XML Schema type and a java type) but what type for the 'Price' element?

While processing the 'Price' element, Castor is going to process the type of 'Price' i.e. the simpleType 'PriceType' which base is 'decimal'. Since derived types are automatically mapped to parent types and W3C XML Schema 'decimal' type is mapped to a `java.math.BigDecimal`, the price element will be a `java.math.BigDecimal`.

Another private variable is created for 'quantity': quantity is mapped to a primitive java type so a `boolean 'has_quantity'` is created for monitoring the state of the quantity variable.

The rest of the code is the 'getter-setter' methods and the Marshalling framework specific methods.

Here the whole Item class:

```
/*
 * This class was automatically generated with
 * <a href="http://castor.exolab.org">Castor 0.8.10</a>, using an
 * XML Schema.
 * $Id$
 */

package test;

  //---------------------------------/
 //- Imported classes and packages -/
//---------------------------------/

import java.io.Reader;
import java.io.Serializable;
import java.io.Writer;
import java.math.BigDecimal;
import org.exolab.castor.xml.*;
import org.exolab.castor.xml.MarshalException;
import org.exolab.castor.xml.ValidationException;
import org.xml.sax.DocumentHandler;

/**
 *
 * @version $Revision$ $Date$
**/
public class Item implements java.io.Serializable {


      //--------------------------/
     //- Class/Member Variables -/
    //--------------------------/

    private java.lang.String _id;
    private int _quantity;

    /**
     * keeps track of state for field: _quantity
    **/
    private boolean _has_quantity;

    private java.math.BigDecimal _price;


      //----------------/
     //- Constructors -/
    //----------------/

    public Item() {
        super();
    } //-- test.Item()
      //-----------/
     //- Methods -/
    //-----------/

    /**
    **/
    public java.lang.String getId()
    {
        return this._id;
    } //-- java.lang.String getId()

    /**
    **/
    public java.math.BigDecimal getPrice()
    {
        return this._price;
    } //-- java.math.BigDecimal getPrice()

    /**
    **/
    public int getQuantity()
    {
        return this._quantity;
    } //-- int getQuantity()

    /**
    **/
    public boolean hasQuantity()
    {
        return this._has_quantity;
    } //-- boolean hasQuantity()
```

```
    /**
    **/
    public boolean isValid()
    {
        try {
            validate();
        }
        catch (org.exolab.castor.xml.ValidationException vex) {
            return false;
        }
        return true;
    } //-- boolean isValid()


    /**
     *
     * @param out
    **/
    public void marshal(java.io.Writer out)
        throws
org.exolab.castor.xml.MarshalException,org.exolab.castor.xml.ValidationException
   {


        Marshaller.marshal(this, out);
    } //-- void marshal(java.io.Writer)


    /**
     *
     * @param handler
    **/
    public void marshal(org.xml.sax.DocumentHandler handler)
        throws org.exolab.castor.xml.MarshalException,
org.exolab.castor.xml.ValidationException
    {
        Marshaller.marshal(this, handler);
    } //-- void marshal(org.xml.sax.DocumentHandler)


    /**
     *
     * @param _id
    **/
    public void setId(java.lang.String _id)
    {
        this._id = _id;
    } //-- void setId(java.lang.String)


    /**
     *
     * @param _price
    **/

public void setPrice(java.math.BigDecimal _price)
    {
        this._price = _price;
    } //-- void setPrice(java.math.BigDecimal)


    /**
     *
     * @param _quantity
    **/
    public void setQuantity(int _quantity)
    {
        this._quantity = _quantity;
        this._has_quantity = true;
    } //-- void setQuantity(int)


    /**
     *
     * @param reader
    **/
    public static test.Item unmarshal(java.io.Reader reader)
        throws
org.exolab.castor.xml.MarshalException,org.exolab.castor.xml.ValidationException
   {
       return (test.Item) Unmarshaller.unmarshal(test.Item.class, reader);
   } //-- test.Item unmarshal(java.io.Reader)


    /**
    **/
    public void validate()
        throws org.exolab.castor.xml.ValidationException
    {
        org.exolab.castor.xml.Validator.validate(this, null);
    } //-- void validate()


}
```

The ItemDescriptor class is a bit more complex. This class is containing inner classes which are the XML field descriptors for the different components of an ‘Item’ element i.e. id, quantity and price.

4 FAQ

- I have many simpleTypes in my schema but there is no classes generated to represent these simpleTypes?

A simpleType remains merely a type. It has no particular meaning in the structure of an XML Schema or an instance if an XML Schema.

Right now the Source Generator generates source for a simpleType only when this simpleType is an enumeration.

- How does the Source Generator handle the attribute ‘maxOccurs’ ?

If you use in your schema an element with the attribute ‘maxOccurs’ set to a number greater than 1, the Source Generator will create a collection that can contain many elements. This collection depends on the property set up in the castorbuilder.properties file. For more details on collections, you can refer to section 2.2.1.

For instance, if you use the Source Generator with the default property and you want generate source for the following element:

```
<element name="Value" type="integer" maxOccurs="unbounded"/>
```

The result will be

```
Private java.util.Vector ValueList
```

with some specific methods for dealing with the Vector:

```
addValue(int vValue),java.util.Enumeration enumerateValue(),
int   getValue(int index), int[] getValue(),
int getValueCount(),removeAllValue(),
int removeValue(int index),
setValue(int vVlaue, int index), setValue(int[] ValueArray).
```

Note: an array setter method is ***automatically*** created.

For example, given the following schema fragment:

```
…
  <complexType>
        <element name=“products” type=“product” minOccurs=“0” maxOccurs= “unbounded”>
  </complexType>
…
```

The source code generator will produce the following “array setter” method:

```
…
   public void setProducts(Product[] productArray) {
      …
   }
…
```

- What is the compatibility with the different JDKs?

The Source Generator supports the different version of the JDKs since version 1.1.

But some features can work only a Java 2 Platform.

- I want to use the Source Generator with a Castor Schema Object Model not with a data file, does the API allow that?

You can create a new instance of the Source Generator and simply call the method `generateSource(Schema schema, String packageName)` of the `SourceGenerator` class.

- I want to use another implementation of the SAX Parser but Castor seems to rely on Xerces, is there a way to specify another parser?

Castor is not relying on the SAX Parser implementation of Xerces. You can use another implementation by setting the following property in the castor.properties file:

```
# Defines the default XML parser to be used by castor
# The parser must implement org.xml.sax.Parser
#
org.exolab.castor.parser=org.apache.xerces.parsers.SAXParser
```

- Is it possible to generate comments in the generated source ?

If you want to insert comments on a class or element, you can use the ANNOTATION tag in the XML Schema.

The Source Generator is generating javadoc-style comments for all the contents of an ANNOTATION tag.

For instance in the example of section 3, the following:

```
<annotation>
      <documentation>
           This is a test XML Schema for Castor XML.
      </documentation>
 </annotation>
```

generates:

```
/**
 *
 * A simple representation of an invoice
 *
 * @version $Revision$ $Date$
**/
```

- How can I retrieve the content of an element when I used *`mixed='true'`* in the type definition of this element?

In several cases you might want to deal with the content of an element. For instance when using the `mixed` attribute in a complexType or when having this kind of element declaration :

```
<element name="foo">
    <complexType>
        <simpleContent>
             <extension base="integer">
                  <attribute name="bar" type="string"/>
              </extension>
           </simpleContent>
     </complexType>
</element>
```

This schema fragment represents this instance element:

```
<foo bar='test'>123456</foo>
```

It might be useful to retrieve the content which in this particular case is an `integer`.

The Source Generator handles it by generating a content member as well as the getter-setter methods for this member.

In the generated class for the Foo element of the previous element you should have:

```
/**
 * internal content storage
**/
 private int _content;


 /**
  * keeps track of state for field: _content
 **/
 private boolean _has_content;


    /**
    **/
    public void deleteContent()
    {
        this._has_content= false;
    } //-- void deleteContent()


   /**
    **/
    public int getContent()
    {
        return this._content;
    } //-- int getContent()


    /**
    **/
    public boolean hasContent()
    {
        return this._has_content;
    } //-- boolean hasContent()
    /**
     *
     * @param _content
    **/
    public void setContent(int _content)
    {
        this._content = _content;
        this._has_content = true;
    } //-- void setContent(int)
```

Note: Due to its special role, the name ‘content’ is a reserved name in Castor and should not be used in a schema.

- XML naming conventions allows the use of several characters that are not allowed in Java members (for instance ‘.’ or ‘:’). How the Source Generator is treating these members?

When encountering such members the Source Generator turns their names by using its own naming convention.

The following example sums up these conventions:

```
<element name="aName" type="string"/>

<element name="Another Name" type="string"/>

<element name="Another.Name" type="string"/>

<element name="last:name" type="string"/>

<element name="last-name" type="string"/>

<element name="last_name" type="string"/>
```

The Source Generator will generate the following Java members:

```
String _aName;

String _anotherName;

String _lastName;
```

- How does the SourceGenerator handles the element *`<any>`*?

XML Schema provides a mechanism that allows to include any well-formed XML fragment in an XML instance document: the element <any>.

Each time <any> is used in an XML Schema, the Source Code Generator creates a Vector that can receive any XML well-formed fragment mapped into a Castor AnyNode.

A Castor AnyNode (`org.exolab.castor.types.AnyNode`) is an alternative to DOM to store well-formed XML fragment. It is directly based on Xpath Node.

The API allows to manipulate the XML fragment and adapters from SAX to an AnyNode and from an AnyNode to SAX are provided (see `org.exolab.castor.xml.util.AnyNode2SAX` and `org.exolab.castor.xml.util.SAX2ANY`).

5 Open Issues: possible improvements

- Propose a Fine grained control on each class
- Provide a target (in Ant) so that we can compile directly the generated classes
- Provide an XML input file to control some properties:

    - Choose the name of the generated files
    - Decide what is the super class of the generated class
    - Decide what is the type of the generated classes (abstract, final, private…)

- Plug with a GUI schema editor
- Generate java classes for the JDO sides

6 Glossary

DOM (Document Object Model)

Document Object Model provides a standard set of objects for representing and manipulating HTML and XML documents.

SAX (Simple API for XML)

SAX is a standard interface for event-based XML parsing.

XML (Extensible Markup Language)

The Extensible Markup Language (XML) is a subset of SGML. Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML.

XML Schema

An XML Schema is a specific XML language that describes the structure and the types of an XML document.

XML Data binding

Representing an XML document directly in-memory.

Marshalling Framework

The marshalling framework is responsible for doing the conversion between Java and XML.

7 References

W3C XML SCHEMA

XML Schema Part 1: Structures

XML Schema Part 2: Datatypes

W3C Candidate Recommendation 24 October 2000

See http://www.w3.org/TR/2000/CR-xmlschema-1-20001024

http://www.w3.org/TR/2000/CR-xmlschema-2-20001024

SAX 2.0

Simple API for XML Version 2.0

David Megginson et al. 5th May 2000

See http://www.megginson.com/SAX/

XPath

XML Path Language

W3C Recommendation 16th November 1999

See http://www.w3.org/TR/xpath/

Sun XML Data Binding WhitePaper

An XML Data Binding Facility for the Java Platform.

Sun Microsystems, Inc. 30 July 1999.

See http://java.sun.com/xml/docs/bind.pdf

Castor XML

Castor XML.

The Exolab group.

See http://castor.exolab.org