
SISE

A simple serializer from Java to XML

Bruno Ranschaert, S.D.I.-Consulting BVBA

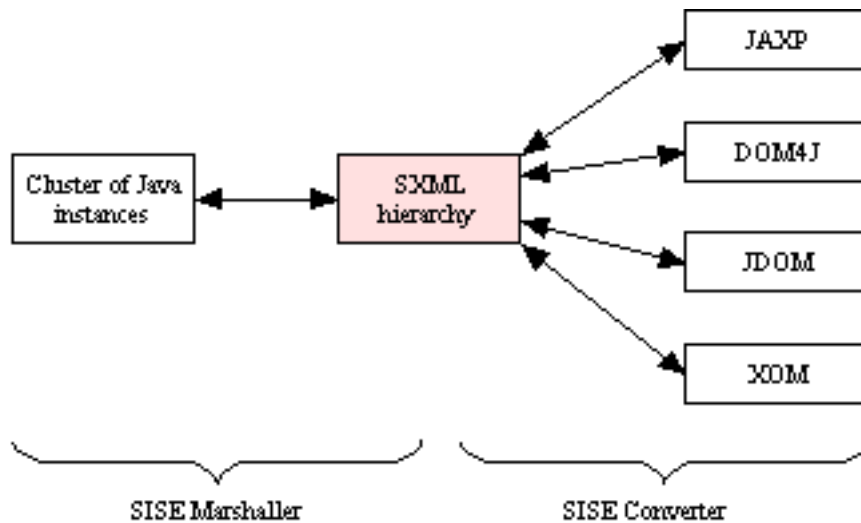
Table of Contents

1. Introduction	1
1.1. Description	1
1.2. Why use SISE?	2
1.3. Goal	2
1.4. License	3
2. Quick Start	3
2.1. Serializing a JavaBean	3
2.2. Convert SXML to XML	4
3. SXML	5
3.1. Primitive Types	5
3.2. Reference Types	5
4. Building the project	7
5. Reference	8
5.1. General	8
5.2. Alternative XML serialization frameworks	8
5.3. Complete XML frameworks	8

1. Introduction

1.1. Description

SISE is a Java library that allows you to convert a cluster of Java objects ("POJO's") to a well defined XML format and vice versa. It works similar to the standard Java serialization mechanism, the format is a subset of XML instead of a binary stream. The focus is on the representation of Java data. Other frameworks focus on the XML and provide mechanisms to represent all XML concepts into Java. SISE works the other way around. SISE wants to represent Java objects in a subset XML called "Simple XML" or SXML. The SXML subset should be as simple as possible, simple enough to represent all Java data correctly. Furthermore, SISE needs to be extendable so that the user of the framework can choose how instances of a class are represented.



This figure shows two important components, we have the Marshaller which converts the POJO's to SXML and we have the specific converters which convert the SXML representation into a general XML representation.

1.2. Why use SISE?

- A Java application can keep its data in standard serialized form. An export function could create the predefined SXML form. Standard XML tools can be used to convert the XML data (e.g. XSLT). Data of an old version of the application can be transformed into data suitable for a newer version.
- Translation from an external XML into a Java object model. Using XSLT.
- A fast way to create an XML protocol for a web service. You can model the protocol using Java-Beans and render them to XML easily.

1.3. Goal

It is the goal of the SISE framework:

- Represent a medium sized cluster of Java POJO's into SXML.
- SXML should be a well defined XML subset.
- The system must be extendable so that the representation can be influenced by the user of the framework.
- Provide interfaces to existing XML frameworks.

Note that SISE does not want to be a general XML framework. As a consequence:

- We will not attempt to do XML parsing, we leave this to the existing libraries.
- We will only envisage DOM systems, this library will not be suited for very large clusters of objects that cannot be kept in memory at once.

1.4. License

SISE - Simple Framework to Serialize JavaBeans to/from a subset of XML.
Copyright (C) 2002, 2005 S.D.I.-Consulting BVBA
<http://www.sdi-consulting.com>
<mailto://nospam@sdi-consulting.com>

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

2. Quick Start

2.1. Serializing a JavaBean

We first define a JavaBean class, we added several interesting properties. Note that we also added a reference to another instance (see the ptr property).

```
public class MyBean
{
    private int id;
    private String name;
    private MyBean ptr;

    private Integer int1, int2;

    public int getId()
    {
        return id;
    }

    public void setId(int aId)
    {
        id = aId;
    }

    public String getName()
    {
        return name;
    }

    public void setName(String aName)
    {
        name = aName;
    }

    public MyBean getPtr()
```

```

    {
        return ptr;
    }

    public void setPtr(MyBean aPtr)
    {
        ptr = aPtr;
    }

    public Integer getInt1()
    {
        return int1;
    }

    public void setInt1(Integer aInt1)
    {
        int1 = aInt1;
    }

    public Integer getInt2()
    {
        return int2;
    }

    public void setInt2(Integer aInt2)
    {
        int2 = aInt2;
    }
}

```

Now we can write some SISE code.

```

❶ MyBean lTest1 = new MyBean();
lTest1.setId(100);
lTest1.setName("SISE Rules!");
lTest1.setInt1(new Integer(0));
lTest1.setInt2(new Integer(0));
❷ Marshall marshall = new MarshallImpl();
❸ MarshallValue lResult = marshall.unmarshall(marshall.marshall(lTest1));
❹ MyBean lTest2 = (MyBean) lResult.getReference();

```

- ❶ A MyBean instance is created, we put it in lTest1. We also initialize the properties.
- ❷ A SISE marshaller is created.
- ❸ We marshall our instance and we immediately unmarshall it. The result is a MarshallValue. We get this extra return type because normally we don't know in advance whether we will receive a reference type or a primitive value. This extra level of indirection allows us to investigate the type before fetching the actual value.
- ❹ Finally we extract the unmarshalled reference from the MarshallValue. The lTest2 instance now contains a clone of the lTest1 instance.

2.2. Convert SXML to XML

We will use DOM4J as our XML engine in the next example. The goal of the next example is to show how you can use full blown XML engines in combination with SXML.

```

❶ Marshall lRenderer = new MarshallImpl();
❷ MyBean lMyBean = new MyBean();

```

```
lMyBean.setId(1003);  
lMyBean.setName("This is a test...");  
❸ lMyBean.setPtr(lMyBean);  
❹ Element lElement = lRenderer.marshall(lMyBean);  
❺ XMLWriter lWriter = new XMLWriter(System.out, OutputFormat.createPrettyPrint());  
❻ lWriter.write(Dom4jConverter.toDom4jDocument(lElement));
```

- ❶ A SISE marshaller is created.
- ❷ Create a JavaBean instance; initialize the properties.
- ❸ We add a circular reference, it can be whatever bean, this only makes the example more interesting.
- ❹ We convert the JavaBean into SXML.
- ❺ We initialize a DOM4J writer, note that the writer is part of the DOM4J library, it is independent of SISE.
- ❻ Finally we use the Dom4jConverter to convert the SXML into DOM4J general XML, and we pretty print it using the DOM4J framework.

3. SXML

SXML stands for "Simple XML". Why use a subset of XML? The answer is that we do not need all the XML possibilities. SISE wants to be simple and understandable. What are the major differences between SXML and XML?

- There is no "document" concept in SXML. Only "elements" and "attributes".
- An "element" can contain text and child elements, but not a mix of these two. An element can have both text and child elements, but they cannot be intertwined, the two are separate concepts.
para>

Since it is one of SISE's goal to generate predictable XML, we will describe how all Java data structures are rendered in the following paragraphs.

3.1. Primitive Types

SISE distinguishes primitive types and reference types. An Integer object will not be mixed with an int representation. If a primitive type is serialized, a primitive type will be deserialized as well. The XML of a primitive type looks like this:

```
<pri type="int"❶ value="101"❷ />
```

- ❶ The type can be: byte, char, short, int, boolean, long, float, double.
- ❷ The value representation is obtained by using the corresponding classes: Byte, Character, Short, Integer, Boolean, Long, Float, Double.

3.2. Reference Types

3.2.1. null

Null is simply serialized as:

```
<null/>
```

3.2.2. Arrays

Arrays are handled in a special way. Java arrays are similar to Object instances, but they are a different concept. They are created and handled differently in the Java virtual machine.

```
<arr class="I" ❶ >
  <pri type="int" value="0"/> ❷
  <pri type="int" value="1"/>
  <pri type="int" value="2"/>
  <pri type="int" value="3"/>
  <pri type="int" value="4"/>
  <pri type="int" value="5"/>
  <pri type="int" value="6"/>
  <pri type="int" value="7"/>
  <pri type="int" value="8"/>
  <pri type="int" value="9"/>
</arr>
```

- ❶ The class contains an indication of the type of the array elements. It is obtained from the class name from the classes from the array objects. If the array contains primitive elements, then the class attribute will contain the type indicator, if the array contains arrays, the class attribute will contain a "[" prefix, and if the array contains other reference types then the class attribute will contain the fully qualified package name of the content class.
- ❷ The elements of the array are rendered recursively, the same system is applied here.

3.2.3. Objects

Objects which are not arrays are serialized using a tree of helpers. Each helper knows how to serialize a specific class. The helpers are organized into a tree, according to the inheritance tree of the classes they handle.

As a result, the serializer finds the best matching helper for an object. There are a number of helpers installed by default, one of these is the `ObjectHelper`, it handles the `Object` class. This class is used if there is no better match available for an object.

Helpers are needed for classes which cannot be serialized using the default `ObjectHelper`. Some built in classes do not conform to the `JavaBean` standard, but they should be serializable.

3.2.3.1. ObjectHelper

The `ObjectHelper` looks for getter and setter methods using introspection. During serialization, only the properties that have a getter and a setter method will be serialized recursively. The serializer is not completely `JavaBean` compliant, because it does not support indexed properties at this time. If you have a class:

```
public class TestBean
{
  private int id;
  private String name;
  private TestBean ptr;
  private Integer int1, int2;

  // Getters and setters omitted.
```

```
}
```

The serialized form will look like:

```
< obj id="id0"❶ class="sise.TestBean"❷ >
  <prop name="name">
    <obj id="id1" class="java.lang.String">This is a test...</obj>
  </prop>
  <prop name="int2">❸
    <null/>
  </prop>
  <prop name="id">
    <obj id="id2" class="java.lang.Integer">1003</obj>
  </prop>
  <prop name="ptr">
    <obj-ref ref="id0"/>❹
  </prop>
  <prop name="int1">
    <null/>
  </prop>
</obj>
```

- ❶ The id contains a unique identifier for each new object that hasn't been serialized before.
- ❷ The class contains the fully qualified class name of the object being serialized.
- ❸ For each property found, there is a `<prop name="..." />` element containing the name of the property. The name is derived from the getter and will be used to deduce the setter method. The contents of the prop element is obtained recursively.
- ❹ If an object has been serialized before, it is serialized as a `<obj-ref ref="..." />` element. In the example, there is a property named `ptr` containing a reference to itself.

3.2.3.2. Helpers for basic type classes

An instance of the `Integer` class is serialized using the `IntegerHelper` as:

```
<obj id="id2" class="java.lang.Integer">1003</obj>
```

Note that the helpers have access to the body of the element, in this case "1003", but not to the surrounding `obj` element. The helpers do not influence the format. The basic type classes are: `Byte`, `Character`, `Short`, `Integer`, `Boolean`, `Long`, `Float`, `Double`, `String`

3.2.3.3. DateHelper

A format was chosen so that no information would get lost. An example date:

```
<obj id="id0" class="java.util.Date">2003-09-20 13:46:56,488 CEST</obj>
```

4. Building the project

The project is built using Jakarta Maven 1.0.2. You first have to install a working version of Maven on your development machine. You can find it on the Jakarta website. The docbook manual can be rendered to HTML or to PDF using the `sdocbook` plugin which can be found in the `maven-plugins` project on SourceForge. I will not describe in detail how this should be done, consult the Maven manu-

als on how to do this.

I experienced a problem while automatically installing the sdocbook plugin. When running the maven sdocbook command the system complained that the jimi-1.0.jar was missing. This jar used to be available in the Maven repository but due to license problems (it is from Sun) the jar had to be removed. You should download the jimi library yourself from Sun, extract the JimiProClasses.zip from the download, rename it to jimi-1.0.jar and put in your local repository {yourhome}/.maven/repository/jimi/jars/jimi-1.0.jar

A second problem when downloading the plugin on Linux is that you should have the rights to write the plugin into the installation directory of Maven. Consult your system administrator when in doubt.

5. Reference

5.1. General

Maven	A build system for Java applications. It is similar to Ant, but it takes another approach. It starts out with a predefined project model (which can be adjusted) and the plugins are aware of this model, so the plugins can be smarter. More information on the website.
Docbook , sdocbook	An XML format that defines how books and articles could be described in XML. More information about Docbook can be found here. I use the maven-sdocbook plugin to generate the manual. More information about the plugin can be found here.

5.2. Alternative XML serialization frameworks

XMLEncoder, XMLDecoder	There is a similar mechanism available in Java since version 1.4. SISE was used since 2000. The advantage of SISE is its simplicity and the complete availability of the source. You can tweak SISE anyway you want.
---------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

5.3. Complete XML frameworks

JAXP	The Java API for XML Processing (JAXP) enables applications to parse and transform XML documents independent of a particular XML processing implementation.
DOM4J	dom4j is an easy to use, open source library for working with XML, XPath and XSLT on the Java platform using the Java Collections Framework and with full support for DOM, SAX and JAXP.
JDOM	A complete, Java-based solution for accessing, manipulating, and outputting XML data from Java code.
XOM	XOM is the only XML API that makes no compromises on correctness. XOM only accepts namespace well-formed XML documents, and only allows you to create namespace well-formed XML documents.