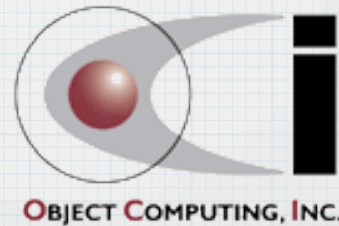




<http://xstream.codehaus.org/>

R. Mark Volkman
mark@ociweb.com
March 2009



Features

- * Uses reflection to simplify
 - * serializing Java objects to XML
 - * deserializing XML to Java objects
- * Fast
- * Minimal memory usage
 - * can output large XML documents
- * Supports object graphs and circular references
- * Customizable
 - * when reflection alone doesn't provide desired XML output
- * Can also produce non-XML output such as JSON
- * Works with Java 1.3 and later

It's simple when the relationships between Java objects matches the relationships between desired XML elements and output requires little or no customization.

Address.java

```
package com.ociweb.data;

public class Address {
    private String city;
    private String state;
    private String street;
    private String zip;

    public Address(
        String street, String city,
        String state, String zip) {
        this.street = street;
        this.city = city;
        this.state = state;
        this.zip = zip;
    }

    public String getCity() { return city; }
    public String getState() { return state; }
    public String getStreet() { return street; }
    public String getZip() { return zip; }

    public void setCity(String city) {
        this.city = city;
    }
    public void setState(String state) {
        this.state = state;
    }
    public void setStreet(String street) {
        this.street = street;
    }
    public void setZip(String zip) {
        this.zip = zip;
    }
}
```

Note that fields are declared in sorted order.

Person.java

```
package com.ociweb.data;

import java.util.Calendar;

public class Person {
    private Address address;
    private Calendar birthdate;
    private String name;

    public Person(
        String name,
        Calendar birthdate,
        Address address) {
        this.address = address;
        this.birthdate = birthdate;
        this.name = name;
    }

    public Address getAddress() { return address; }
    public Calendar getBirthdate() { return birthdate; }
    public String getName() { return name; }

    public void setAddress(Address address) {
        this.address = address;
    }
    public void setBirthdate(Calendar birthdate) {
        this.birthdate = birthdate;
    }
    public void setName(String name) {
        this.name = name;
    }
}
```

Note that fields are declared in sorted order.

Main.java

```
import com.ocicweb.data.Address;
import com.ocicweb.data.Person;
import com.thoughtworks.xstream.XStream;
import java.util.GregorianCalendar;

public class Main {

    public static void main(String[] args) {
        Address address = new Address(
            "123 Some Street", "Some City", "MO", "12345");
        Person person = new Person(
            "Mark Volkmann", new GregorianCalendar(3, 16, 1961), address);

        XStream xstream = new XStream();
        String xml = xstream.toXML(person);
        System.out.println(xml);
    }
}
```

Easy!

Output

```
<com.ocicweb.data.Person>
  <address>
    <city>Some City</city>
    <state>MO</state>
    <street>123 Some Street</street>
    <zip>12345</zip>
  </address>
  <birthdate>
    <time>-61861341600000</time>
    <timezone>America/Chicago</timezone>
  </birthdate>
  <name>Mark Volkmann</name>
</com.ocicweb.data.Person>
```

Issues:

1. don't want root element name to be full class name
2. birthdate content isn't human readable
3. want birthdate to be an attribute
4. child elements aren't in preferred order, they are in field declaration order

Customizing Element Names

- * Can specify "aliases" for Java classes that are used for corresponding element names

```
xstream.alias("person", Person.class);
```

- * Output is now

```
<person>  
  . . .  
</person>
```

Custom Converters ...

- * Converters are Java classes that convert specific types of Java objects to and from XML
- * Two types
 - * implement **Converter** interface to convert a Java object to an XML element
 - * implement **SingleValueConverter** interface to convert a Java object to a single value that can be the value of an attribute or the text inside an element
- * Example
 - * we'll create a converter for **Calendar** objects that converts them to a single value

... Custom Converters ...

```
package com.ociweb.xstream;  
  
import com.thoughtworks.xstream.converters.ConversionException;  
import com.thoughtworks.xstream.converters.SingleValueConverter;  
import java.text.DateFormat;  
import java.text.ParseException;  
import java.text.SimpleDateFormat;  
import java.util.Calendar;  
import java.util.Date;  
import java.util.GregorianCalendar;  
  
public class CalendarConverter implements SingleValueConverter {  
  
    public boolean canConvert(Class clazz) {  
        return Calendar.class.isAssignableFrom(clazz);  
    }  
}
```

testing whether clazz
is the Calendar class
or a subclass of it

... Custom Converters

```
public Object fromString(String value) {  
    DateFormat formatter = new SimpleDateFormat("M/d/yyyy");  
    try {  
        Date date = formatter.parse(value);  
        Calendar cal = new GregorianCalendar();  
        cal.setTime(date);  
        return cal;  
    } catch (ParseException e) {  
        String msg = "can't parse \"" + value + "\" into a Date";  
        throw new ConversionException(msg, e);  
    }  
}  
  
public String toString(Object obj) {  
    DateFormat formatter = new SimpleDateFormat("M/d/yyyy");  
    Calendar cal = (Calendar) obj;  
    return formatter.format(cal.getTime());  
}
```

don't have to test type before casting
since this won't be called
unless canConvert returns true

Custom Converters ...

- * Can register it to be used for all **Calendar** values

```
xstream.registerConverter(new CalendarConverter());
```

- * Can register it to only be used for specific fields

```
xstream.registerLocalConverter(  
    Person.class, "birthdate",  
    new CalendarConverter());
```

Using Attributes

- * Fields are serialized as child elements by default
- * To serialize a specific field as attributes instead

```
xstream.useAttributeFor(Person.class, "birthdate");
```

- * Output is now

```
<person birthdate="4/16/1961">  
  <address>  
    <city>Some City</city>  
    <state>MO</state>  
    <street>123 Some Street</street>  
    <zip>12345</zip>  
  </address>  
  <name>Mark Volkmann</name>  
</person>
```

CalendarConverter
was used to produce
this attribute value.

Field Order ...

- * By default, child elements are output in the order in which their corresponding fields are declared
- * Customize with a **FieldKeySorter**

```
SortableFieldKeySorter sorter = new SortableFieldKeySorter();  
  
sorter.registerFieldOrder(  
    Person.class, new String[] {"name", "birthdate", "address"});  
  
sorter.registerFieldOrder(  
    Address.class, new String[] {  
        "street", "city", "state", "zip"});  
  
XStream xstream = new XStream(  
    new PureJavaReflectionProvider(new FieldDictionary(sorter)));
```

need to include birthdate in list even though it is being output as an attribute; will get StreamException otherwise

... Field Order

- * Output is now

```
<person birthdate="4/16/1961">  
  <name>Mark Volkmann</name>  
  <address>  
    <street>123 Some Street</street>  
    <city>Some City</city>  
    <state>MO</state>  
    <zip>12345</zip>  
  </address>  
</person>
```

Omitting Fields

* Three ways to configure

1. make field transient

* ex. `private transient String zip;`

2. use `omitField` method

* ex. `xstream.omitField(Address.class, "zip");`

3. annotate field with `@XStreamOmitField`

* not detected automatically; need

`xstream.processAnnotations(ClassName.class);`

or

`xstream.autodetectAnnotations(true);`

* see <http://xstream.codehaus.org/annotations-tutorial.html#AutoDetect>

Duplicate References ...

* When multiple objects share an object reference ...

```
Address address = new Address (
    "123 Some Street", "Some City", "MO", "12345");
Person person1 = new Person (
    "Mark Volkmann", new GregorianCalendar(1961, 3, 16), address);
Person person2 = new Person (
    "Tami Volkmann", new GregorianCalendar(1961, 9, 9), address);

// All the previous customization goes here.

List<Person> people = new ArrayList<Person>();
people.add(person1);
people.add(person2);
xstream.alias("people", List.class);
String xml = xstream.toXML(people);
System.out.println(xml);
```


... Duplicate References

* Output

```
<people>
  <person birthdate="4/16/1961">
    <address>
      <city>Some City</city>
      <state>MO</state>
      <street>123 Some Street</street>
      <zip>12345</zip>
    </address>
    <name>Mark Volkmann</name>
  </person>
  <person birthdate="10/9/1961">
    <address reference="../../person/address"/>
    <name>Tami Volkmann</name>
  </person>
</people>
```

"reference" attribute values are relative XPath expressions for finding a shared reference.

Stream On!

- * BSD licensed
- * Project Founder: Joe Walnes (software engineer at Google)
- * Project Lead: Jörg Schaible
- * <http://xstream.codehaus.org/> contains
 - * downloads
 - * link to API documentation (javadoc)
 - * tutorials
 - * architecture overview

The latest release was version 1.3.1 on 12/8/08.

Easier With WAX? ...

```
import com.ocweb.data.Address;
import com.ocweb.data.Person;
import com.ocweb.xml.WAX;
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.GregorianCalendar;

public class Main {
    private WAX wax = new WAX();

    public static void main(String[] args) {
        new Main();
    }

    private Main() {
        Address address = new Address("123 Some Street", "Some City", "MO", "12345");
        Person person = new Person("Mark Volkmann", new GregorianCalendar(1961, 3, 16), address);
        toXML(person);
        wax.close();
    }
}
```

A more explicit approach that doesn't use reflection like XStream

... Easier With WAX?

```
private void toXML(Person person) {
    DateFormat formatter = new SimpleDateFormat("M/d/yyyy");
    String birthdate = formatter.format(person.getBirthdate().getTime());

    wax.start("person")
        .attr("birthdate", birthdate)
        .child("name", person.getName());
    toXML(person.getAddress());
    wax.end();
}

private void toXML(Address address) {
    wax.start("street")
        .child("street", address.getStreet())
        .child("city", address.getCity())
        .child("state", address.getState())
        .child("zip", address.getZip())
        .end();
}
}
```