

La sérialisation XML facile avec l'API XStream

par [Eric Reboisson](#)

Date de publication : 1/10/2006

Dernière mise à jour : 1/10/2006

Cet article présente une découverte de l'API XStream pour sérialiser et désérialiser des objets Java dans des fichiers XML.

- I - Introduction
- II - O  trouver XStream ?
- III - Cr ation des classes   s rialiser
- IV - S rialisation des classes
- V - D s rialisation des classes
- VI - Un peu plus...
 - VI-A - Comment emp cher la s rialisation d'un attribut ?
 - VI-B - La s rialisation de collections
 - VI-C - Comment cr er des alias pour les noms de package ?
- VII - XStream : un outil de persistance ?
- VIII - Conclusion
- IX - Remerciements

I - Introduction

XStream est une [API Java](#) qui permet de s rialiser et d s rialiser des objets dans des fichiers [XML](#).

Les avantages d'XStream sont principalement :

- La facilit  d'utilisation de l'[API](#).
- Pas de modification de code des objets que vous voulez s rialiser.
- La rapidit  d'ex cution et une faible utilisation de la m moire.

Toutes les sources de cet article sont disponibles [ici](#) sous la forme d'un projet pour [Eclipse](#).

II - Où trouver XStream ?

XStream est disponible à l'adresse suivante <http://xstream.codehaus.org>.

XStream est téléchargeable sous la forme d'une librairie **JAR** par le menu "Download" du site.

Pour la suite de l'article, nous utiliserons la dernière version stable **xstream-x.x.jar** obtenue sur le site ([Télécharger ici](#)).

La librairie **xstream-x.x.jar** est à mettre dans le [classpath](#) pour que les exemples fonctionnent.

III - Cr ation des classes   s rialiser

Nous allons premi rement cr er deux classes, **Entete** et **Article**.

La classe **Article** d clarera un attribut de classe de type **Entete**.

Ces deux classes contiennent les constructeurs, getters et setters utiles.

La classe **Entete** :

Cr ation de la classe Entete

```
package beans;

import java.util.Date;

public class Entete {

    private String titre;

    private Date dateCreation;

    public Date getDateCreation() {
        return dateCreation;
    }

    public void setDateCreation(Date dateCreation) {
        this.dateCreation = dateCreation;
    }

    public String getTitre() {
        return titre;
    }

    public void setTitre(String titre) {
        this.titre = titre;
    }

    public Entete(String titre, Date dateCreation) {
        super();
        this.titre = titre;
        this.dateCreation = dateCreation;
    }

}
```

Et la classe **Article** qui utilise la classe **Entete** :

Cr ation de la classe Article

```
package beans;

public class Article {

    private Entete entete;

    private String synopsis;

    public String getSynopsis() {
        return synopsis;
    }

    public Article(Entete entete, String synopsis) {
        super();
        this.entete = entete;
        this.synopsis = synopsis;
    }

}
```

Cr ation de la classe Article

```
    }  
  
    public void setSynopsis(String synopsis) {  
        this.synopsis = synopsis;  
    }  
  
    public Entete getEntete() {  
        return entete;  
    }  
  
    public void setEntete(Entete entete) {  
        this.entete = entete;  
    }  
}
```

IV - S rialisation des classes

Ci-dessous une classe **Serialisation** avec une m thode *main* contenant le code pour convertir les classes pr c dentes en cha ne XML :

Un main pour tester la s rialisation

```
package tests;
import java.util.Date;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.xml.DomDriver;

public class Serialisation {

    public static void main(String[] args) {
        // Instanciation de la classe XStream
        XStream xstream = new XStream(new DomDriver());
        // Instanciation de la classe Entete
        Entete entete = new Entete("Titre de l'article",new Date());
        // Instanciation de la classe Article
        Article article = new Article(entete,"Un synopsis bien plac  !!! <strong>avec une
balise HTML</strong>");
        // Conversion du contenu de l'objet article en XML
        String xml = xstream.toXML(article);
        // Affichage de la conversion XML
        System.out.println(xml);
    }
}
```

La console affichera :

Le contenu de l'objet article convertit en XML

```
<beans.Article>
  <entete>
    <titre>Titre de l'&apos;article</titre>
    <dateCreation>2006-10-01 18:00:31.187 CEST</dateCreation>
  </entete>
  <synopsis>Un synopsis bien plac  !!! &lt;strong&gt;avec une balise HTML&lt;/strong&gt;</synopsis>
</beans.Article>
```

XStream convertit le contenu de l'objet **article** en un joli fichier XML, on remarquera que les caract res sp ciaux sont  galement trait s :

- ' en '

- en

Maintenant, pour s rialiser l'objet **article** en un fichier *article.xml*, le code de la classe serait :

S rialisation de l'objet article dans un fichier article.xml

```
package tests;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Date;

import beans.Article;
import beans.Entete;
```

S rialisation de l'objet article dans un fichier article.xml

```
import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.xml.DomDriver;

public class Serialisation {

    public static void main(String[] args) {

        try {
            // Instanciacion de la classe XStream
            XStream xstream = new XStream(new DomDriver());
            // Instanciacion de la classe Entete
            Entete entete = new Entete("Titre de l'article", new Date());
            // Instanciacion de la classe Article
            Article article = new Article(entete, "Un synopsis bien plac  !!! <strong>avec une
balise HTML</strong>");

            // Instanciacion d'un fichier c:/temp/article.xml
            File fichier = new File("c:/temp/article.xml");
            // Instanciacion d'un flux de sortie fichier vers
            // c:/temp/article.xml
            FileOutputStream fos = new FileOutputStream(fichier);
            try {
                // S rialisation de l'objet article dans c:/temp/article.xml
                xstream.toXML(article, fos);
            } finally {
                // On s'assure de fermer le flux quoi qu'il arrive
                fos.close();
            }
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
}
```

V - Désérialisation des classes

Maintenant pour recharger (désérialiser) un objet de type Article avec les données sérialisées dans le chapitre précédent, le code sera :

Désérialisation du fichier article.xml vers un objet de type Article

```
package tests;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;

import beans.Article;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.xml.DomDriver;

public class Deserialisation {

    public static void main(String[] args) {

        try {
            // Instanciation de la classe XStream
            XStream xstream = new XStream(new DomDriver());

            // Redirection du fichier c:/temp/article.xml vers un flux
            // d'entrée fichier
            FileInputStream fis = new FileInputStream(new File("c:/temp/article.xml"));

            try {
                // Désérialisation du fichier c:/temp/article.xml vers un nouvel
                // objet article
                Article nouvelArticle = (Article) xstream.fromXML(fis);

                // Affichage sur la console du contenu de l'attribut synopsis
                System.out.println(nouvelArticle.getSynopsis());

            } finally {
                // On s'assure de fermer le flux quoi qu'il arrive
                fis.close();
            }

        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
}
```

VI - Un peu plus...

Dans ce chapitre, quelques informations et exemples pour aller plus loin avec l'API XStream.

VI-A - Comment empêcher la sérialisation d'un attribut ?

En déclarant un attribut comme *transient*, celui-ci ne sera pas sérialisé.

Dans l'exemple suivant l'attribut *numeroCb* ne sera donc pas sérialisé :

Empêcher la sérialisation d'un attribut

```
package beans;

public class Auteur {

    private String nom;

    private String prenom;

    private transient String numeroCb;

    //...constructeurs, setters et getters
}
```

VI-B - La sérialisation de collections

La sérialisation de collections avec XStream se fait très facilement, prenons pour exemple une classe **Magasin** contenant une liste d'objets :

Un attribut de type List à sérialiser

```
package beans;

import java.util.ArrayList;
import java.util.List;

public class Magasin {
    private String nom;

    // Supprimer les "<Object>" si vous utilisez un jdk < 1.5
    List<Object> objets = new ArrayList<Object>();

    //...constructeurs, setters et getters
}
```

Et le code suivant, qui peuplera la classe **Magasin** avec des objets de type **Article** et **Auteur** :

Sérialisation de collection

```
package tests;

import java.util.ArrayList;
import java.util.Date;
import java.util.List;

import beans.Article;
import beans.Auteur;
import beans.Entete;
import beans.Magasin;

import com.thoughtworks.xstream.XStream;
import com.thoughtworks.xstream.io.xml.DomDriver;
```

S rialisation de collection

```

public class CollectionSerialisation {

    public static void main(String[] args) {
        // Instanciation de la classe XStream
        XStream xstream = new XStream(new DomDriver());
        // Instanciation de la classe Entete
        Entete entete = new Entete("Titre de l'article", new Date());
        // Instanciation de la classe Article
        Article article = new Article(entete, "Un synopsis bien plac  !!! <strong>avec une balise
HTML</strong>");
        // Instanciation de la classe Auteur
        Auteur auteur = new Auteur("LA LA LA LA LA LA", "Starsky & Hutch");
        auteur.setNumerocb("1788 1803 0485 1875");

        // Instanciation de la classe Magasin
        Magasin magasin = new Magasin("Yatoutici");
        List objets = new ArrayList();
        objets.add(article);
        objets.add(auteur);
        magasin.setObjets(objets);

        // Conversion du contenu de l'objet article en XML
        String xml = xstream.toXML(magasin);
        // Affichage de la conversion XML
        System.out.println(xml);
    }
}

```

La console affichera :

```

<beans.Magasin>
  <nom>Yatoutici</nom>
  <objets>
    <beans.Article>
      <entete>
        <titre>Titre de l'&apos;article</titre>
        <dateCreation>2006-10-01 17:48:48.78 CEST</dateCreation>
      </entete>
      <synopsis>Un synopsis bien plac  !!! &lt;/strong&gt;avec une balise
HTML&lt;/strong&gt;</synopsis>
    </beans.Article>
    <beans.Auteur>
      <nom>LA LA LA LA LA LA</nom>
      <prenom>Starsky & Hutch</prenom>
    </beans.Auteur>
  </objets>
</beans.Magasin>

```

VI-C - Comment cr er des alias pour les noms de package ?

Vous avez d  remarquer qu'XStream utilisait les noms des packages pour nommer les balises XML, ce n'est pas bien lisible.

Heureusement, XStream propose un syst me permettant de cr er des alias sur les classes utilis es lors de la s rialisation.

Ainsi en reprenant notre premier exemple de s rialisation, on cr era les alias de la mani re suivante :

Cr ation d'alias

```

...
        // Conversion du contenu de l'objet article en XML
        xstream.alias("article", Article.class);

```

Cr ation d'alias

```
xstream.alias("entete", Entete.class);
String xml = xstream.toXML(article);
// Affichage de la conversion XML
System.out.println(xml);
...
```

La console affichera les balises substitu es avec les alias :

La balise <beans.Article> convertie en <article> avec les alias

```
<article>
  <entete>
    <titre>Titre de l'apos;article</titre>
    <dateCreation>2006-10-01 18:08:06.234 CEST</dateCreation>
  </entete>
  <synopsis>Un synopsis bien plac  !!! &lt;strong&gt;avec une balise HTML&lt;/strong&gt;</synopsis>
</article>
```

VII - XStream : un outil de persistance ?

A cette question, je répondrai : oui et non ! Une réponse de normand me direz vous, je m'explique :

Alors oui, car XStream permet effectivement d'enregistrer, puis de recouvrer des objets sérialisés, mais cela uniquement en utilisant une sérialisation **XML** (A noter à ce propos qu'XStream propose des classes très utiles comme `XmlArrayList`, `XmlSet`, ou `XmlMap`).

Et non, parce que j'associe persistance de données avec **SGBD** et ça XStream ne sait pas faire. Pour cela il faudra utiliser des outils comme Hibernate, Castor, JPox, iBatis, OJB, etc. et certains de ces outils savent utiliser les deux formes de stockage (**SGBD** ou **XML**) pour la persistance.

Mais si vous optez pour une persistance **XML** avec XStream alors sachez :

- Qu'aucune description n'est nécessaire pour sérialiser la plupart des objets (pas de "mapping")
- XStream utilise l'introspection pour retrouver les propriétés d'un bean, elles pourront être déclarées private, ne pas avoir de getters ou setters et XStream retrouvera ses petits !

VIII - Conclusion

A l'utilisation d'XStream, on remarque très vite que cette [API](#) va rendre de nombreux services.

XStream est une librairie idéale pour les néophytes du DOM et autre SAX, ils trouveront plus de facilités dans la création et les manipulations de fichiers XML en utilisant XStream.

Et les finalités d'XStream sont très nombreuses, le transport de données, utilisation de fichiers de configuration, etc.

Pour exemple d'utilisation, j'ai préconisé XStream pour transporter toute une configuration paramétrée dans une base de données vers une application embarquée devant fonctionner en mode déconnecté.

[Télécharger les sources de l'article](#)

IX - Remerciements

Un grand merci   [le y@m's](#) pour la relecture de cet article et   [GrandFather](#) pour sa bonne id e.