Dozer

v.5.5.1

User's Guide

Franz Garsombke, Matt Tierney, Dmitry Buzdin

2014-04-22

Table of Contents

1. **Table of Contents** i
2. **Why Map?** 1
3. **Getting Started** 2
4. **Usage** 5
5. **Mappings via Annotations** 8
6. **Mappings via API** 10
7. **Mappings via XML** 12
7..1. Mapping Two Classes 14
7..2. Basic Property Mapping 15
7..3. String To Date Mapping 17
7..4. Enum Mapping 19
7..5. Collection Mapping 21
7..6. Map Backed Properties 27
7..7. Index Mapping 32
7..8. Deep Mapping 33
7..9. Excluding Fields 36
7..10. One-Way Mapping 39
7..11. Context Based Mapping 41
7..12. Global Configuration 43
7..13. Custom Converters 46
7..14. Custom Bean Factories 55
7..15. Custom Create Methods 58
7..16. Custom Get/Set Methods 59
7..17. Copy By Ref Mapping 61
7..18. Inheritance Mapping 63
7..19. Dozer Events 67
7..20. Expression Language 69
8. **Configuration** 70
8..1. Logging 72
8..2. Proxy Handling 73
9. **Statistics** 74
10. **JMX Integration** 75
11. **Spring Integration** 77
12. **JAXB and XMLBeans** 79
13. **Metadata Query Interface** 81
14. **FAQ** 83

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

1 Why Map?

1.1 Why Map?

A mapping framework is useful in a layered architecture where you are creating layers of abstraction by encapsulating changes to particular data objects vs. propagating these objects to other layers (i.e. external service data objects, domain objects, data transfer objects, internal service data objects). A mapping framework is ideal for using within Mapper type classes that are responsible for mapping data from one data object to another.

For distributed systems, a side effect is the passing of domain objects between different systems. Typically, you won't want internal domain objects exposed externally and won't allow for external domain objects to bleed into your system.

Mapping between data objects has been traditionally addressed by hand coding value object assemblers (or converters) that copy data between the objects. Most programmers will develop some sort of custom mapping framework and spend countless hours and thousands of lines of code mapping to and from their different data object.

A generic mapping framework solves these problems. Dozer is an open source mapping framework that is robust, generic, flexible, reusable, and configurable.

Data object mapping is an important part of layered service oriented architectures. Pick and choose the layers you use mapping carefully. Do not go overboard as there is maintenance and performance costs associated with mapping data objects between layers.

1.1.1 Parallel Object Hierarchies

There could be different reasons of why application should support parallel object hierarhchies. To name a few:

- Integration with External Code
- Serialization Requirements
- Framework Integration
- Separation of Architectural Layers

In some cases it is efficient to guard your code base from frequently changing object hierarchy, which you do not control directly. In this case Dozer serves as a bridge between application and external objects. As mapping is performed in reflective manner not all changes break your API. For example if object changes from Number to String the code will keep working as this is resolved automatically.

Some frameworks impose Serialization constraints, which does not allow sending any Java objects through the wire. One of the popular examples is Google Web Toolkit (GWT) framework, which restricts developer to sending only objects compiled to JavaScript and marked as Serializable. Dozer helps to convert Rich Domain Model to Presentation Model, which satisfies GWT serialization requirements.

Dozer integrates nicely with frameworks such as Apache XmlBeans and JAXB implementations. With help of provided factory classes, conversion between domain model and Xml objects is defined in the same way as plain object to object mappings.

In complex enterprise application it is often valuable to separate design to several architectural layers. Each of them would reside on its own abstraction level. A typical simplified example would be presentation, domain and persistence layers. Each of this layers could have own set of Java Beans representing data relevant for this layer. It is not necessary that all data will travel to the upper levels of architecture. For example the same domain object could have different mappings dependant of the presentation layer requirements.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

2 Getting Started

2.1 Getting Started

2.1.1 Downloading the Distribution

- Download Dozer and extract the archive.
- Add dozer.jar to your classpath.
- Add required thirdparty runtime jars to your classpath.

If you are using Maven, simply copy-paste this dependency to your project.

```
<dependency>
    <groupId>net.sf.dozer</groupId>
    <artifactId>dozer</artifactId>
    <version>5.4.0</version>
</dependency>
```

Apache Ivy users can copy-paste the following line instead.

```
<dependency org="net.sf.dozer" name="dozer" rev="5.4.0"/>
```

2.1.2 1st Mapping

For your first mapping lets assume that the two data objects share all common attribute names.

```
Mapper mapper = new DozerBeanMapper();
DestinationObject destObject =
    mapper.map(sourceObject, DestinationObject.class);
```

After performing the Dozer mapping, the result will be a new instance of the destination object that contains values for all fields that have the same field name as the source object. If any of the mapped attributes are of different data types, the Dozer mapping engine will automatically perform data type conversion. At this point you have completed your first Dozer mapping. Later sections will go over how to specify custom mappings via custom xml files.

IMPORTANT: For real-world applications it is NOT recommended to create a new instance of the Mapper each time you map objects. Typically a system will only have one DozerBeanMapper instance per VM. If you are not using an IOC framework where you can define the Mapper Instance as singleton="true". If needed, a DozerBeanMapperSingletonWrapper convenience class has been provided in the Dozer jar.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

2.1.3 Specifying Custom Mappings via XML

If the two different types of data objects that you are mapping contain any fields that don't share a common property name, you will need to add a class mapping entry to your custom mapping xml file. These mappings xml files are used at runtime by the Dozer mapping engine.

Dozer automatically performs any type conversion when copying the source field data to the destination field. The Dozer mapping engine is bi-directional, so if you wanted to map the destination object to the source object, you do not need to add another class mapping to the xml file.

IMPORTANT: Fields that are of the same name do not need to be specified in the mapping xml file. Dozer automatically maps all fields with the same property name from the source object into the destination object.

```
<mapping>
  <class-a>yourpackage.yourSourceClassName</class-a>
  <class-b>yourpackage.yourDestinationClassName</class-b>
    <field>
      <a>yourSourceFieldName</a>
      <b>yourDestinationFieldName</b>
    </field>
</mapping>
```

The complete Dozer mapping xml file would look like the following. The Custom Mappings section contains more information on mapping options that are available to you for more complex use cases.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">

  <configuration>
    <stop-on-errors>true</stop-on-errors>
    <date-format>MM/dd/yyyy HH:mm</date-format>
    <wildcard>true</wildcard>
  </configuration>

  <mapping>
    <class-a>yourpackage.yourSourceClassName</class-a>
    <class-b>yourpackage.yourDestinationClassName</class-b>
      <field>
        <A>yourSourceFieldName</A>
        <B>yourDestinationFieldName</B>
      </field>
  </mapping>

  other custom class mappings would go here.......

</mappings>
```

2.1.4 Dozer and Dependency Injection Frameworks

Dozer is not dependant of any existing Dependency Injection framework (DI). However the general aim is to support the most typical use cases with ready-to-use Wrappers. Check Spring Integration manual for option of initializing Dozer in context of Spring DI framework.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

3 Usage

3.1 General Usage

Requirements for running Dozer:

- Dozer uses SLF4J for logging purposes.
- Dozer needs a few third-party runtime jars.
- All of the required runtime jars are in the {dozer.home}/repository directory and need to be in your Classpath
- The dozer.jar file in the {dozer.home}/dist directory needs to be in your Classpath

3.1.1 Dozer Bean Mapper

Before we go over setting up custom xml bean mappings, let us look at a simple example of using Dozer. The Dozer mapping implementation has a method called map which takes a source object and either a destination object or destination object class type. After mapping the two objects it then returns the destination object with all of its mapped fields.

```
Mapper mapper = new DozerBeanMapper();

DestinationObject destObject =
    mapper.map(sourceObject, DestinationObject.class);
            or
DestinationObject destObject = new DestinationObject();
mapper.map(sourceObject, destObject);
```

IMPORTANT: For real-world applications it is not recommended that you create a new instance of the Mapper each time you map objects. Typically a system will only have one DozerBeanMapper instance per VM. If you are not using an IOC framework where you can define the Mapper as singleton="true", a DozerBeanMapperSingletonWrapper convenience class has been provided in the Dozer jar.

Dozer operates in two general modes: implicit and explicit. Implicit mode is activated by default and tries to resolve mappings for you. It uses simple assumptions that if two objects are passed for mapping then bean properties with the same names should be mapped. If there are additional mappings needed, which can not be derived by the naming you should add those either via Xml, Annotations or API.

Explicit mode assumes that no mappings should be performed or "guessed" until you have specified those specifically. The amount of coding is higher in explicit mode, but sometimes you would like to have full control on what is going on during the mappings process and this approach is also used in many of the productive applications. Implicit/Explicit mapping switch is called *"wildcard"* in Dozer. Whenever you encounter that in configuration you know what behavior to expect from now on.

3.1.2 Injecting Custom Mapping Files

The Dozer mapping xml file(s) define any custom mappings that can't be automatically performed by the Dozer mapping engine. Any custom Dozer mapping files need to be injected into the Mapper

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

implementation(org.dozer.DozerBeanMapper). Both setter-based and constructor-based injection are supported.

Preferably, you will be using an IOC framework such as Spring for these Dozer injection requirements. Alternatively, the injection of mapping files can be done programmatically. Below is a programmatic approach to creating a bean mapper. Note that this is **NOT the recommended way to retrieve the bean mapper** . Each new instance needs to be initialized and this consumes time as well as resources. If you are using the mapper this way just wrap it using the singleton pattern.

```
List myMappingFiles = new ArrayList();
myMappingFiles.add("dozerBeanMapping.xml");
myMappingFiles.add("someOtherDozerBeanMappings.xml");
DozerBeanMapper mapper = new DozerBeanMapper();
mapper.setMappingFiles(myMappingFiles);
DestinationObject destObject =
    mapper.map(sourceObject, DestinationObject.class);
```

IMPORTANT: Mapper instance(s) should be setup as a Singleton regardless of how you choose to inject the Mapper instance(s). You should configure the Mapper this way so that you do not have to reload and reinitialize the mapping files for each individual mapping during the lifecycle of your app. Reinitializing the mapping files for each mapping would be inefficient and unnecessary. The DozerBeanMapper.java class is thread safe.

3.1.3 Spring Integration

The following is an example how the Mapper bean would be configured via Spring.
Sample spring.xml bean definition...

```
<bean id="mapper" class="org.dozer.DozerBeanMapper">
  <property name="mappingFiles">
    <list>
      <value>dozer-global-configuration.xml</value>
      <value>dozer-bean-mappings.xml</value>
      <value>more-dozer-bean-mappings.xml</value>
    </list>
  </property>
</bean>
```

3.1.4 Dozer Bean Mapper Singleton Wrapper

There is one way to configure the DozerBeanMapperSingletonWrapper to use your custom mapping file.

- **Using one mapping file:** A file called dozerBeanMapping.xml file will be loaded if it is in your Classpath. You can find a sample of this file in the {dozer.home}/mappings directory.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

The mapping file defines all of the relationships between Java classes and their attributes. The Custom Mappings section details the custom XML mapping options that are available.

The following example show how to use the DozerBeanMapperSingletonWrapper. Dozer has a method called map which takes a source object and either a destination object or destination object class type. After mapping the two objects it then returns the destination object with all of its mapped fields.

```
Mapper mapper = DozerBeanMapperSingletonWrapper.getInstance();
DestinationObject destObject =
    mapper.map(sourceObject, DestinationObject.class);

or

Mapper mapper = DozerBeanMapperSingletonWrapper.getInstance();
DestinationObject destObject = new DestinationObject();
mapper.map(sourceObject, destObject);
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

4 Mappings via Annotations

4.1 Annotation Mappings

4.1.1 Why Annotations?

One of the downsides of using Dozer for the long time was Xml. Since Dozer started during Xml-hype years more than five years ago that was pretty obvious choice back then. After that Java 5 brought us annotations and new industry accepted style of configuring behaviour are Domain-Specific Languages. DSL-like support is provided in form of mapping API, but since version 5.3.2 Dozer starts providing annotations support as well.

The obvious reasons to use annotations is to avoid duplicating field and method names in your mapping code. The annotation can be put onto the mapped property itself thus reducing the amount of code. However there are cases when annotations should be avoided or even impossible to use. Some of them are the following:

- You are mapping classes, which are not under your control, but provided in libraries;
- The mappings are quite complex and require many configurations;

In the first case you could be mapping JAXB generated entities or third-party DTOs and have no possibility to put annotations. In the second case there is a choice of putting lots of multi-line annotations or isolating the mapping code with certain duplication of entity names. Overannotated beans could be problematic to read and understand.

4.1.2 Usage

WARNING: Annotation support in Dozer is experimental and does not cover complex use cases yet. However it may be useful to implement that simplest mappings you have had to do in Xml or API before.

The idea is very simple. You put @Mapping annotation either on getter of field directly. If Dozer finds that it adds a bi-directional mapping. It means that putting annotation once will create mappings for both conversion types. Type conversions (e.g. String-Long) will be chosen automatically. Global custom converters are resolved as well. Annotations work only if a conversion is subject to wildcard rule (active by default). The following example demonstrates annotations in action.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
public class SourceBean {

    private Long id;

    private String name;

    @Mapping("binaryData")
    private String data;

    @Mapping("pk")
    public Long getId() {
        return this.id;
    }

    public String getName() {
        return this.name;
    }
}
```

```
public class TargetBean {

    private String pk;

    private String name;

    private String binaryData;

    public void setPk(String pk) {
        this.pk = pk;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

Mapping the given beans with Dozer will result in all three fields being mapped. Property "name" will be mapped by naming convention. Property "id" will be transformed to "pk". Field "data" will be moved to "binaryData". Do not worry about *private* modifier; it will be handled automatically.

Currently Dozer offers only one annotation, but the next ones will be added in following releases. As for now you can mix and match all flavours of mapping types to achieve the desired effect: Xml, API and Annotations.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

5 Mappings via API

5.1 API Mappings

Since version 5.3 Dozer offers two ways for specifying explicit mappings. First is Xml-based and has been there for years. Second is API-based and it is brand new. On the high level both approaches are functionally equivalent, however there are major differences in using those described further on.

5.1.1 XML Mapping Flaws

XML-based approach is stable and is used in many production projects. However, it imposes several limitations.

- First, and most important, is that it can not be dynamically generated. All XML mappings should be present on Dozer start-up and can not be modified afterwards. There are tricky ways, when you can generate and put mappings to the file system by your own templating engine, but this approach is not supported by Dozer itself. By generating custom mappings you are able to automate repetitive chunks of low-level Dozer language.
- Second problem is that you are forced to duplicate all of your Bean class names in Xml mappings. This leads to lots of typing and copy-paste programming. This can be partly compensated by use of Expression Language inside Xml, but it is not solving all of the problems.
- Refactoring support is limited as IDE should keep track of class names in Xml files and change them when you rename or move the referenced class. Auto-completion support is also not available in all IDEs.

5.1.2 API Mappings

API mappings are intended to solve all of the mentioned problems. To preserve backwards compatibility API mappings can be combined with existing Xml mappings. In fact some parts of the configuration (e.g. global configuration block) are only possible to express in Xml format.

To get a feeling of what are these mappings take a look at the following code example.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
import static org.dozer.loader.api.FieldsMappingOptions.*;
import static org.dozer.loader.api.TypeMappingOptions.*;

...

BeanMappingBuilder builder = new BeanMappingBuilder() {
      protected void configure() {
        mapping(Bean.class, Bean.class,
                TypeMappingOptions.oneWay(),
                mapId("A"),
                mapNull(true)
        )
                .exclude("excluded")
                .fields("src", "dest",
                        copyByReference(),
                        collectionStrategy(true,
                            RelationshipType.NON_CUMULATIVE),
                        hintA(String.class),
                        hintB(Integer.class),
                        FieldsMappingOptions.oneWay(),
                        useMapId("A"),
                        customConverterId("id")
                )
                .fields("src", "dest",
                    customConverter("org.dozer.CustomConverter")
                );
      }
    };
```

Constructed builder object should be then passed to DozerBeanMapper instance. It is possible to add multiple Builder classes.

```
DozerBeanMapper mapper = new DozerBeanMapper();
mapper.addMapping(builder);
```

Don't forget to make a static import of FieldsMappingOptions and TypeMappingOptions classes.

That's it.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

6 Mappings via XML

6.1 Custom Mappings Via Dozer XML Files

This section will cover setting up custom mappings in xml file(s). If the two different types of data objects that you are mapping contain any fields that don't share a common property name, you will need to add a class mapping entry to your custom mapping xml file. These mappings xml files are used at runtime by the Dozer mapping engine.

Dozer automatically performs any type conversion when copying the source field data to the destination field. The Dozer mapping engine is bi-directional, so if you wanted to map the destination object to the source object, you do not need to add another class mapping to the xml file.

An example of a mapping file....

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">

  <mapping>
    <class-a>org.dozer.vo.TestObject</class-a>
    <class-b>org.dozer.vo.TestObjectPrime</class-b>
    <field>
      <a>one</a>
      <b>onePrime</b>
    </field>
  </mapping>

  <mapping wildcard="false">
    <class-a>org.dozer.vo.TestObjectFoo</class-a>
    <class-b>org.dozer.vo.TestObjectFooPrime</class-b>
      <field>
        <a>oneFoo</a>
        <b>oneFooPrime</b>
      </field>
  </mapping>

</mappings>
```

A mappings element has multiple mapping elements, each with class mapping declarations and field level mappings. The wildcard attribute is set to true by default. This means that it will automatically try to map every property in the two objects. When the attribute is set to false it will only map explicitly defined fields.

IMPORTANT: Properties that are of the same name do not need to be specified in the mapping xml file. Dozer automatically maps all fields with the same property name from the source object into the destination object.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

6.2 How Custom Mapping Files Are Loaded

Dozer will search the entire classpath looking for the specified file. The generally acceptable way of distributing your mappings is to bundle them inside your application archive.

Alternatively, you can load files from outside the classpath by prepending "file:" to the resource name. Ex) "file:c:\somedozermapping.xml"

6.3 Loading Files from Input Stream

Since version 5.4.0 it is possible to load XML mapping files from provided InputStream object. Check DozerMapper class for the corresponding API calls.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

7 Mapping Two Classes

7.1 Mapping Classes

An example of mapping two classes is defined below. Note: Explicit xml mapping for 2 classes is not required if all the field mapping between src and dest object can be performed by matching on attribute name. Custom xml class mapping is only required when you need to specify any custom field mappings.

```
<mappings>
  <mapping>
    <class-a>org.dozer.vo.TestObject</class-a>
    <class-b>org.dozer.vo.TestObjectPrime</class-b>
    <!-- Any custom field mapping xml would go here -->
  </mapping>
</mappings>
```

These mappings are bi-directional so you would never need to define an XML map for TestObjectPrime to TestObject. If these two classes had references to complex types that needed type transformation, you would also define them as mappings. Dozer recursively goes through an object and maps everything in it. Data type conversion is performed automatically. Dozer also supports no attribute mappings at all. If supplied two classes that are not mapped, it simply tries to map properties that are the same name.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

8 Basic Property Mapping

8.1 Basic Property Mapping

8.1.1 Implicit Property Mapping (bi-directional)

Matching field names are automatically handled by Dozer.

Properties that are of the same name do not need to be specified in the mapping xml file.

8.1.2 Simple Mappings (bi-directional)

We will start off simple. If you have two properties with different names they can be mapped as such:

```
<field>
  <a>one</a>
  <b>onePrime</b>
</field>
```

8.1.3 Data type conversion

Data type coversion is performed automatically by the Dozer mapping engine. Currently, Dozer supports the following types of conversions: (these are all bi-directional)

- Primitive to Primitive Wrapper
- Primitive to Custom Wrapper
- Primitive Wrapper to Primitive Wrapper
- Primitive to Primitive
- Complex Type to Complex Type
- String to Primitive
- String to Primitive Wrapper
- String to Complex Type if the Complex Type contains a String constructor
- String to Map
- Collection to Collection
- Collection to Array
- Map to Complex Type
- Map to Custom Map Type
- Enum to Enum
- Each of these can be mapped to one another: java.util.Date, java.sql.Date, java.sql.Time, java.sql.Timestamp, java.util.Calendar, java.util.GregorianCalendar
- String to any of the supported Date/Calendar Objects.
- Objects containing a toString() method that produces a long representing time in (ms) to any supported Date/Calendar object.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

8.1.4 Recursive Mapping (bi-directional)

Dozer supports full Class level mapping recursion. If you have any complex types defined as field level mappings in your object, Dozer will search the mappings file for a Class level mapping between the two Classes that you have mapped. If you do not have any mappings, it will only map fields that are of the same name between the complex types.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

9 String To Date Mapping

9.1 String to Date Mapping

A date format for the String can be specified at the field level so that the necessary data type conversion can be performed.

```
<field>
  <a date-format="MM/dd/yyyy HH:mm:ss:SS">dateString</a>
  <b>dateObject</b>
</field>
```

A default date format can also be specified at the class mapping level. This default date format will be applied to all field mappings unless it is overridden at the field level.

```
<mapping date-format="MM-dd-yyyy HH:mm:ss">
  <class-a>org.dozer.vo.TestObject</class-a>
  <class-b>org.dozer.vo.TestObjectPrime</class-b>
  <field>
    <a>dateString</a>
    <b>dateObject</b>
  </field>
</mapping>
```

A default date format can also be specified at the very top mappings level. This default date format will be applied to all field mapppings unless it is overridden at a lower level

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<mappings>
  <configuration>
    <date-format>MM/dd/yyyy HH:mm</date-format>
  </configuration>

  <mapping wildcard="true">
    <class-a>org.dozer.vo.TestObject</class-a>
    <class-b>org.dozer.vo.TestObjectPrime</class-b>
    <field>
      <a>dateString</a>
      <b>dateObject</b>
    </field>
  </mapping>
  <mapping>
    <class-a>org.dozer.vo.SomeObject</class-a>
    <class-b>org.dozer.vo.SomeOtherObject</class-b>
    <field>
      <a>srcField</a>
      <b>destField</b>
    </field>
  </mapping>
</mappings>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

10 Enum Mapping

10.1 JDK 1.5 Enum Mapping

To map an enums value to another enum is shown below.

```
<field>
  <a>status</a>
  <b>statusPrime</b>
</field>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
enum Status {
  PROCESSING, SUCCESS, ERROR
}

public class UserGroup {

  private Status status;

  public Status getStatus() {
    return status;
  }

  public void setStatus(Status status) {
    this.status = status;
  }

}

enum StatusPrime {
  PROCESSING, SUCCESS, ERROR
}

public class UserGroupPrime {

  private StatusPrime statusPrime;

  public StatusPrime getStatusPrime() {
    return statusPrime;
  }

  public void setStatusPrime(StatusPrime statusPrime) {
    this.statusPrime = statusPrime;
  }

}
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

11 Collection Mapping

11.1 Collection and Array Mapping

Dozer automatically maps between collection types and automatically performs any type conversion. Each element in the source collection is mapped to an element in the dest object. Hints are used to specify what type of objects are created in the destination collection. The following collection mapping is automatically handled by Dozer: (These are all bi-directional)

- List to List
- List to Array
- Array to Array
- Set to Set
- Set to Array
- Set to List

11.1.1 Using Hints for Collection Mapping

Hints are not required if you are using JDK 1.5 Generics or Arrays because the types can be autodetected by Dozer. But if you are not using generics or Arrays, to convert a Collection/Array to a Collection/Array with different type objects you can specify a Hint to let Dozer know what type of objects you want created in the destination list. If a Hint is not specified for the destination field, then the destination Collection will be populated with objects that are the same type as the elements in the src Collection.

```
<!-- converting TheFirstSubClass List to
     TheFirstSubClassPrime List -->
<field>
  <a>hintList</a>
  <b>hintList</b>
  <b-hint>org.dozer.vo.TheFirstSubClassPrime</b-hint>
</field>
```

Below is a summary of the mapping logic used when mapping Arrays, Sets, and Lists. This gives a breakdown of what happens when hints are or are not used.

- List to List
-
  - Dest Hint req'd: NO
  - Dest Hint allowed: YES
  - If no dest hint specified: Dest list will contain the same data types in the source
  - If hint is speficied: Dest list will contain objects that match dest hint(s) type
- Array to List
-
  - Dest Hint req'd: NO
  - Dest Hint allowed: YES
  - If no dest hint specified: Dest list will contain the same data types in the source
  - If hint is speficied: Dest list will contain objects that match dest hint(s) type

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

- List to Array
- - Dest Hint req'd: NO
  - Dest Hint allowed: YES
  - If no dest hint specified: Dest array will contain data types defined by the array
  - If hint is speficied: Dest list will contain objects that match dest hint(s) type (only if Object Array)
- Array to Array
- - Dest Hint req'd: NO
  - Dest Hint allowed: YES
  - If no dest hint specified: Dest array will contain data types defined by the array
  - If hint is speficied: Dest list will contain objects that match dest hint(s) type (only if Object Array)
- Set to Set
- - Dest Hint req'd: NO
  - Dest Hint allowed: YES
  - If no dest hint specified: Dest list will contain the same data types in the source
  - If hint is speficied: Dest list will contain objects that match dest hint(s) type
- Array to Set
- - Dest Hint req'd: NO
  - Dest Hint allowed: YES
  - If no dest hint specified: Dest list will contain the same data types in the source
  - If hint is speficied: Dest list will contain objects that match dest hint(s) type
- Set to Array
- - Dest Hint req'd: NO
  - Dest Hint allowed: YES
  - If no dest hint specified: Dest array will contain data types defined by the array
  - If hint is speficied: Dest list will contain objects that match dest hint(s) type (only if Object Array)
- List to Set
- - Dest Hint req'd: NO
  - Dest Hint allowed: YES
  - If no dest hint specified: Dest list will contain the same data types in the source
  - If hint is speficied: Dest list will contain objects that match dest hint(s) type
- Set to List
- - Dest Hint req'd: NO
  - Dest Hint allowed: YES
  - If no dest hint specified: Dest list will contain the same data types in the source
  - If hint is speficied: Dest list will contain objects that match dest hint(s) type

11.1.2 Using JDK 1.5 Generics for Collection Mapping

Hints are not required when JDK 1.5 Generics are used. To convert a Collection/Array to a Collection/Array with different type objects dozer can determine parameterized types at runtime.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
public class UserGroup {

  private Set<User> users;

  public Set<User> getUsers() {
    return users;
  }

  public void setUsers(Set<User> aUsers) {
    users = aUsers;
  }

}
public class UserGroupPrime {

  private List<UserPrime> users;

  public List<UserPrime> getUsers() {
    return users;
  }

  public void setUsers(List<UserPrime> aUsers) {
    users = aUsers;
  }

}
```

11.1.3 Object Array to List (bi-directional)

When converting an Object array to a List, by default the destination List will contain the same data type as the source Array.

```
<!-- changing an Integer [] to List and back again -->
<field>
  <a>arrayForLists</a>
  <b>listForArray</b>
</field>
```

Use a hint for data type conversion. Because a hint is specified, the destination List will contain String elements instead of Integers.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<!-- changing an Integer [] to List and back again -->
<field>
  <a>arrayForLists</a>
  <b>listForArray</b>
  <b-hint>java.lang.String</b-hint>
</field>
```

11.1.4 Primitive Array to Primitive Array (bi-directional)

When converting an Object array to an Array, by default the destination Array will contain the same data type as the source Array.

```
<!-- converting int[] to int [] by name only -->
<field>
  <a>anArray</a>
  <b>theMappedArray</b>
</field>
```

11.1.5 Cumulative vs. Non-Cumulative List Mapping (bi-directional)

If you are mapping to a Class which has already been initialized, dozer will either 'add' or 'update' objects to your List. If your List or Set already has objects in it dozer checks the mapped List, Set, or Array and calls the contains() method to determine if it needs to 'add' or 'update'. This is determined using the relationship-type attribute on the field tag. The default is 'cumulative'. relationship-type can be specifed at the field mapping, class mapping, or global configuration level.

global configuration level....

```
<mappings>
  <configuration>
     <relationship-type>non-cumulative</relationship-type>
  </configuration>
</mappings>
```

class mapping level....

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<mappings>
  <mapping relationship-type="non-cumulative">
    <class-a>org.dozer.vo.TestObject</class-a>
    <class-b>org.dozer.vo.TestObjectPrime</class-b>
    <field>
      <a>someList</a>
      <b>someList</b>
    </field>
  </mapping>
</mappings>
```

field mapping level....

```
<!-- objects will always be added to an existing List -->
<field relationship-type="cumulative">
  <a>hintList</a>
  <b>hintList</b>
  <a-hint>org.dozer.vo.TheFirstSubClass</a-hint>
  <b-hint>org.dozer.vo.TheFirstSubClassPrime</b-hint>
</field>

<!-- objects will updated if already exist in List,
     added if they are not present -->
<field relationship-type="non-cumulative">
  <a>unequalNamedList</a>
  <b>theMappedUnequallyNamedList</b>
</field>
```

Note: if you do not define custom *equals()* and *hashCode()* methods non-cumulative option will not function properly, as Dozer will fail to determine object equality and will rely on JDK generated object Ids. In default case two instances of a class are always treated as different and update will not occure.

11.1.6 Removing Orphans

Orphans are elements which exist in a destination collection that did not exist within the source collection. Dozer will remove orphans by calling the 'remove' method on actual orphans of the underlying destination collection; it will not clear all. To determine elements which are orphans dozer uses the contains() method to check if the results contains orphans. The default setting value is false.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<!-- orphan objects will always be removed from an existing
     destination List -->
<field remove-orphans="true">
  <a>srcList</a>
  <b>destList</b>
</field>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

12 Map Backed Properties

12.1 Map Backed Property Mapping

12.1.1 Map to Map

Dozer will map a java.util.Map to a java.util.Map. If there are complex types with hints it will do deep recursion mapping as well. If the destination map exists it will add elements to the existing map.

```
<mapping>
  <class-a>org.dozer.vo.map.MapToMap</class-a>
  <class-b>org.dozer.vo.map.MapToMapPrime</class-b>

  <field>
    <a>standardMapWithHint</a>
    <b>standardMapWithHint</b>
    <a-hint>org.dozer.vo.TestObject</a-hint>
    <b-hint>org.dozer.vo.TestObjectPrime</b-hint>
  </field>
</mapping>
```

12.1.2 Mapping Field Level Properties to a java.util.Map or a Custom Map with unique Get/Set methods

Dozer supports mapping map backed properties at the field level. The map can either implement the java.util.Map Interface or be a custom map with a set of unique Get/Set methods.

In this example Field A is a basic String and it is mapped to Field B which is a HashMap. The key in the HashMap will be "stringProperty" (the attribute name) and the value will be whatever value is stored in that attribute.

```
<mapping>
  <class-a>org.dozer.vo.map.PropertyToMap</class-a>
  <class-b>org.dozer.vo.map.MapToProperty</class-b>

  <field>
    <a>stringProperty</a>
    <b>hashMap</b>
  </field>
</mapping>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

This example shows Field A is a basic String and it is mapped to Field B which is a HashMap. The key in the HashMap will be "myStringProperty" and the value will be whatever value is stored in that attribute. Also notice that Field A has a unique setter() method name.

```
<mapping>
  <class-a>org.dozer.vo.map.PropertyToMap</class-a>
  <class-b>org.dozer.vo.map.MapToProperty</class-b>

  <field>
    <a set-method="addStringProperty2">stringProperty2</a>
    <b key="myStringProperty">hashMap</b>
  </field>
</mapping>
```

This example shows Field A is a basic String and it is mapped to Field B which is a custom map. The key in the custom map will be "myCustomProperty" and the value will be whatever value is stored in that attribute. Notice that Field B has unique map getter() and map setter() method names. If you are using a custom map you must explicitly set the map Get/Set method names. A destination hint can also be provided if your custom map implements an Interface or is an Abstract class.

```
<mapping>
  <class-a>org.dozer.vo.map.PropertyToMap</class-a>
  <class-b>org.dozer.vo.map.MapToProperty</class-b>

  <field>
    <a>stringProperty3</a>
    <b map-get-method="getValue" map-set-method="putValue"
       key="myCustomProperty">customMap</b>
  </field>

  <field>
    <a>stringProperty4</a>
    <b map-get-method="getValue" map-set-method="putValue"
       key="myCustomNullProperty">nullCustomMap</b>
    <b-hint>org.dozer.vo.map.CustomMap</b-hint>
  </field>

  <field>
    <a>stringProperty5</a>
    <b map-get-method="getValue"
       map-set-method="putValue">customMap</b>
  </field>
</mapping>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

12.1.3 Mapping Class Level Properties to a java.util.Map or a Custom Map with unique Get/Set methods

Dozer can also map entire complex objects directly to a java.util.Map or a custom map object. This example shows the declaration of a mapping between a complex object (PropertyToMap) and a java.util.Map. When doing this you need to explicitly define a unique map-id for the mapping. This is used when determining which map to use at run-time. Every attribute on the PropertyToMap class will be mapped to the java.util.Map. You DO NOT need to explicitly define these mappings. Field exclude mappings can be used to exclude fields at run-time. If the attribute name is not the same as the map key just set the key attribute for a custom field mapping. The mapping to stringProperty2 shows an example of this.

The second example shows how to setup a custom map object. The only difference here is that you need to explicitly define map-set-method and map-get-method values. These correspond to the java.util.Map get() and put() methods.

```
<mapping map-id="myTestMapping">
  <class-a>org.dozer.vo.map.PropertyToMap</class-a>
  <class-b>java.util.Map</class-b>
  <field>
    <a set-method="addStringProperty2">stringProperty2</a>
    <b key="myStringProperty">this</b>
  </field>
  <field-exclude>
    <a>excludeMe</a>
    <b>this</b>
  </field-exclude>

</mapping>

<mapping map-id="myCustomTestMapping">
  <class-a>org.dozer.vo.map.PropertyToMap</class-a>
  <class-b map-set-method="putValue" map-get-method="getValue">
      org.dozer.vo.map.CustomMap
  </class-b>
  <field>
    <a set-method="addStringProperty2">stringProperty2</a>
    <b key="myStringProperty">this</b>
  </field>
  <field-exclude>
    <a>excludeMe</a>
    <b>this</b>
  </field-exclude>

</mapping>
```

The example below shows how to use these mappings. Notice that the field mappings reference a map-id. The first field mapping will use the myTestMapping defined mapping and map accordingly. Same goes with the custom mapping.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<mapping>
  <class-a>org.dozer.vo.map.MapTestObject</class-a>
  <class-b>org.dozer.vo.map.MapTestObjectPrime</class-b>
  <field map-id="myTestMapping">
    <a>propertyToMap</a>
    <b>propertyToMapMap</b>
  </field>

  <field map-id="myTestMapping">
    <a>propertyToMapToNullMap</a>
    <b>nullPropertyToMapMap</b>
    <b-hint>java.util.HashMap</b-hint>
  </field>

  <field map-id="myCustomTestMapping">
    <a>propertyToCustomMap</a>
    <b>propertyToCustomMapMap</b>
  </field>
</mapping>
```

The Class Level map backed mappings can also be used as a standard mapping. For this dozer has a new API. In addition to the source and destination classes you can now pass in the map reference Id.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
// Example 1
PropertyToMap ptm = new PropertyToMap();
ptm.setStringProperty("stringPropertyValue");
ptm.addStringProperty2("stringProperty2Value");
Map map = Mapper.map(ptm, HashMap.class, "myTestMapping");

// Example 2
CustomMap customMap = mapper.map(ptm, CustomMap.class,
    "myCustomTestMapping");

// Example 3
CustomMap custom = new CustomMap();
custom.putValue("myKey", "myValue");
Mapper.map(ptm, custom, "myCustomTestMapping");

// Example 4 - Map Back
Map map = new HashMap();
map.put("stringProperty", "stringPropertyValue");
PropertyToMap property = mapper.map(map, PropertyToMap.class,
    "myTestMapping");
assertEquals("stringPropertyValue", property.getStringProperty());
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

13 Index Mapping

13.1 Indexed Property Mapping

Fields that need to be looked up or written to by indexed property are supported.

```
<mapping>
  <class-a>org.dozer.vo.Individuals</class-a>
  <class-b>org.dozer.vo.FlatIndividual</class-b>
    <field>
      <a>usernames[0]</a>
      <b>username1</b>
    </field>
    <field>
      <a>usernames[1]</a>
      <b>username2</b>
    </field>
    <field>
      <a>individual.username</a>
      <b>username2</b>
    </field>
    <field>
      <a>secondNames[1]</a>
      <b>secondName1</b>
    </field>
    <field>
      <a>secondNames[2]</a>
      <b>secondName2</b>
    </field>
    <field>
      <a>aliases.otherAliases[0]</a>
      <b>primaryAlias</b>
    </field>
</mapping>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

14 Deep Mapping

14.1 Deep Property Mapping

It is possible to map deep properties. An example would be when you have an object with a String property. Your other object has a String property but it is several levels deep within the object graph. In the example below the DestDeepObj has nested attributes within the object graph that need to be mapped. Type hints are supported for deep field mappings. The attributes copy-by-reference, type=one-way, and relationship-type can also be used.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<mapping>
  <class-a>org.dozer.vo.deep.SrcDeepObj</class-a>
  <class-b>org.dozer.vo.deep.DestDeepObj</class-b>

  <field>
    <a>srcNestedObj.src1</a>
    <b>dest1</b>
  </field>

  <field>
    <a>srcNestedObj.src2</a>
    <b>dest2</b>
  </field>

  <field>
    <a>srcNestedObj.srcNestedObj2.src5</a>
    <b>dest5</b>
  </field>

  <field><!-- java.util.List to java.util.List -->
    <a>srcNestedObj.hintList</a>
    <b>hintList</b>
    <a-hint>java.lang.String</a-hint>
    <b-hint>java.lang.Integer</b-hint>
  </field>

  <field>
    <a>srcNestedObj.hintList2</a>
    <b>hintList2</b>
    <a-hint>org.dozer.vo.TheFirstSubClass</a-hint>
    <b-hint>org.dozer.vo.TheFirstSubClassPrime</b-hint>
  </field>

  <field copy-by-reference="true">
    <a>srcNestedObj.hintList3</a>
    <b>hintList3</b>
  </field>

</mapping>
```

14.2 Deep Indexed Mapping

Indexed mapping within deep mapping is supported.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<field>
  <a>offSpringName</a>
  <b>pets[1].offSpring[2].petName</b>
</field>
```

Destination Hints are NOT required if the indexed collection is an Array or if you are using jdk 1.5 Generics. Dozer is able to automatically determine the property type for these use cases. But you will need to provide hints if the data types are not Arrays or if you are not using Generics. This is required so that Dozer knows what types of dest objects to create while it traverses the deep field mapping.

The following is an example of using hints.....

```
<field>
  <a>someField</a>
  <b>someList[1].someOtherList[0].someOtherField</b>
  <b-deep-index-hint>org.dozer.vo.TestObject,
      org.dozer.vo.AnotherTestObject
  </b-deep-index-hint>
</field>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

15 Excluding Fields

15.1 Excluding Fields

Dozer supports excluding fields from a mapping using the field-exclude tag. We also support field excludes going one-way as shown in the example.

```
<field-exclude>
  <a>fieldToExclude</a>
  <b>fieldToExclude</b>
</field-exclude>

<field-exclude type="one-way"">
  <a>fieldToExclude</a>
  <b>fieldToExclude</b>
</field-exclude>
```

15.1.1 Wildcard - excluding default field mappings

There's also a flag (`wildcard`) set on class mapping which controls whether the default mapping (which applies to pair of properties of the same name) should be done. The default value is true. For example:

```
<mapping wildcard="false">
    <class-a>org.dozer.vo.AnotherTestObject</class-a>
    <class-b>org.dozer.vo.AnotherTestObjectPrime</class-b>
    <field>
       <a>field1</a>
       <b>field1</b>
    </field>
</mapping>
```

This configuration would cause only the fields `field1` in both classes to be mapped, even if both classes share a property with the same name called `field2`.

15.1.2 Exclude Mapping Null Values

You can bypass the mapping of null values. If this is specified, the dest field mapping is bypassed at runtime and the destination value setter method will not be called if the src value is null. This can be specified at the mapping or class level. For example:

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<mapping map-null="false">
  <class-a>org.dozer.vo.AnotherTestObject</class-a>
  <class-b>org.dozer.vo.AnotherTestObjectPrime</class-b>
  <field>
    <a>field4</a>
    <b>to.one</b>
  </field>
</mapping>
```

OR...

```
<mapping>
  <class-a>org.dozer.vo.AnotherTestObject</class-a>
  <class-b map-null="false">org.dozer.vo.AnotherTestObjectPrime
  </class-b>
  <field>
    <a>field4</a>
    <b>to.one</b>
  </field>
</mapping>
```

15.1.3 Exclude Mapping Empty Strings

You can bypass the mapping of empty String values. If this is specified, the dest field mapping is bypassed at runtime and the destination value setter method will not be called if the src value is an empty String. This can be specified at the mapping or class level. For example:

```
<mapping map-empty-string="false">
  <class-a>org.dozer.vo.AnotherTestObject</class-a>
  <class-b>org.dozer.vo.AnotherTestObjectPrime</class-b>
  <field>
    <a>field4</a>
    <b>to.one</b>
  </field>
</mapping>
```

OR...

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<mapping>
  <class-a>org.dozer.vo.AnotherTestObject</class-a>
  <class-b map-empty-string="false">
      org.dozer.vo.AnotherTestObjectPrime
  </class-b>
  <field>
    <a>field4</a>
    <b>to.one</b>
  </field>
</mapping>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

16 One-Way Mapping

16.1 One-Way Mapping

You can set how a mapping definition behaves as far as direction goes. If you only want to map two classes to go one-way you can set this at the mapping level. The default is bi-directional. This can be set at the mapping level OR the field level. When one-way is specified, "a" is always the src object and "b" is always the destination object.

```
<mapping type="one-way">
  <class-a>org.dozer.vo.TestObjectFoo</class-a>
  <class-b>org.dozer.vo.TestObjectFooPrime</class-b>
    <field>
      <a>oneFoo</a>
      <b>oneFooPrime</b>
    </field>
</mapping>
```

In the following example the one-way fields are only mapped when "a" object is mapped to "b" object. If "b" is mapped to "a", then the field is not mapped.

```
<mapping>
  <class-a>org.dozer.vo.TestObjectFoo2</class-a>
  <class-b>org.dozer.vo.TestObjectFooPrime2</class-b>
  <field type="one-way">
    <a>oneFoo2</a>
    <b>oneFooPrime2</b>
  </field>

  <field type="one-way">
    <a>oneFoo3.prime</a>
    <b>oneFooPrime3</b>
  </field>
```

16.1.1 Excluding Fields One-Way

Dozer supports field excludes going one-way as shown in the example. In the example the field is only excluded when "a" is mapped to "b". If "b" is mapped to "a", then the field is not excluded.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<field-exclude type="one-way"">
  <a>fieldToExclude</a>
  <b>fieldToExclude</b>
</field-exclude>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

17 Context Based Mapping

17.1 Context Based Mapping

Context based mapping can be specified by using the map-id attribute. Note that we also support nested context mapping by specifying a map-id at the field level.

```
<mapping map-id="caseA">
   <class-a>org.dozer.vo.context.ContextMapping</class-a>
   <class-b>org.dozer.vo.context.ContextMappingPrime</class-b>
   <field-exclude>
     <a>loanNo</a>
     <b>loanNo</b>
   </field-exclude>
   <field map-id="caseC">
     <a>contextList</a>
     <b>contextList</b>
     <b-hint>org.dozer.vo.context.ContextMappingNestedPrime
     </b-hint>
   </field>
</mapping>

<mapping map-id="caseB">
  <class-a>org.dozer.vo.context.ContextMapping</class-a>
  <class-b>org.dozer.vo.context.ContextMappingPrime</class-b>
</mapping>

<mapping map-id="caseC">
  <class-a>org.dozer.vo.context.ContextMappingNested</class-a>
  <class-b>org.dozer.vo.context.ContextMappingNestedPrime
  </class-b>
   <field-exclude>
     <a>loanNo</a>
     <b>loanNo</b>
   </field-exclude>
</mapping>

<mapping map-id="caseD">
  <class-a>org.dozer.vo.context.ContextMappingNested</class-a>
  <class-b>org.dozer.vo.context.ContextMappingNestedPrime
  </class-b>
</mapping>
</mappings>
```

To use a particular context when invoking the Mapper, you simply specify the map-id in your mapping call.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
ContextMappingPrime cmpA =
    mapper.map(cm, ContextMappingPrime.class, "caseA");
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

18 Global Configuration

18.1 Global Configuration

The configuration block is used to set the global default settings. Also, any Custom Converters are defined in this section. The configuration block is entirely "optional".

Dozer supports the ability to have multiple mapping files, but only one global configuration across the multiple mapping files. We recommend having a separate mapping xml file for specifying the single global configuration. **Implicit mappings will inherit the default values for configuration.**

The following is the sample configuration block:

```
<configuration >
  <date-format>MM/dd/yyyy HH:mm</date-format>
  <stop-on-errors>true</stop-on-errors>
  <wildcard>true</wildcard>
  <trim-strings>false</trim-strings>

  <custom-converters> <!-- these are always bi-directional -->
    <converter type="org.dozer.converters.TestCustomConverter" >
      <class-a>org.dozer.vo.TestCustomConverterObject</class-a>
      <class-b>another.type.to.Associate</class-b>
    </converter>

  </custom-converters>
</configuration>
```

18.2 Overriding Wildcards

Each individual mapping section can set its own wildcard policy even if there is a global wildcard policy defined using the configuration block. For example, the following mapping does not allow wildcards:

```
<mapping wildcard="false">
  <class-a>org.dozer.vo.SpringBean</class-a>
  <class-b>org.dozer.vo.SpringBeanPrime</class-b>

  <field>
    <a>anAttributeToMap</a>
    <b>anAttributeToMapPrime</b>
  </field>
</mapping>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

18.3 Overriding Date Format

The same is true for date format values. Each individual mapping section can set its own date format rules. For example:

```
<!-- Override top level date format default  -->
<mapping date-format="MM-dd-yyyy HH:mm:ss">
  <class-a>org.dozer.vo.TestObject</class-a>
  <class-b>org.dozer.vo.TestObjectPrime</class-b>

  <field>
    <a>one</a>
    <b>onePrime</b>
  </field>
</mapping>
```

18.4 Overriding Error Handling

You can override the error handling policy for a particular mapping. For example:

```
<!-- Override top level defaults -->
<mapping stop-on-errors="false">
  <class-a>org.dozer.vo.TestObject</class-a>
  <class-b>org.dozer.vo.TestObjectPrime</class-b>

  <field>
    <a>one</a>
    <b>onePrime</b>
  </field>
</mapping>
```

18.5 Overriding Trim Strings Policy

You can override the trim strings policy for a particular mapping. For example:

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<!-- Override top level defaults -->
<mapping trim-strings="true">
  <class-a>org.dozer.vo.TestObject</class-a>
  <class-b>org.dozer.vo.TestObjectPrime</class-b>

  <field>
    <a>one</a>
    <b>onePrime</b>
  </field>
</mapping>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

19 Custom Converters

19.1 Custom Converters

Custom converters are used to perform custom mapping between two objects. In the Configuration block, you can add some XML to tell Dozer to use a custom converter for certain class A and class B types. When a custom converter is specified for a class A and class B combination, Dozer will invoke the custom converter to perform the data mapping instead of the standard mapping logic.

Your custom converter must implement the org.dozer.CustomConverter interface in order for Dozer to accept it. Otherwise an exception will be thrown.

Custom converters are shared across mapping files. This means that you can define them once in a mapping file and it will be applied to all class mappings in other mapping files that match the Class A - Class B pattern. In the example below, whenever Dozer comes across a mapping where the src/dest class match the custom converter definition, it will invoke the custom converter class instead of performing the typical mapping.

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">
  <configuration>
    <custom-converters> <!-- these are always bi-directional -->
      <converter type="org.dozer.converters.TestCustomConverter" >
        <class-a>org.dozer.vo.CustomDoubleObject</class-a>
        <class-b>java.lang.Double</class-b>
      </converter>

      <!-- You are responsible for mapping everything between
           ClassA and ClassB -->
      <converter
        type="org.dozer.converters.TestCustomHashMapConverter" >
        <class-a>
          org.dozer.vo.TestCustomConverterHashMapObject
        </class-a>
        <class-b>
          org.dozer.vo.TestCustomConverterHashMapPrimeObject
        </class-b>
      </converter>

    </custom-converters>
  </configuration>

</mappings>
```

Custom converters can also be specified at the individual field level. In the example below, Dozer will invoke the custom converter to perform the field mapping.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<mapping>
  <class-a>org.dozer.vo.SimpleObj</class-a>
  <class-b>org.dozer.vo.SimpleObjPrime2</class-b>
  <field custom-converter=
    "org.dozer.converters.StringAppendCustomConverter">
    <a>field1</a>
    <b>field1Prime</b>
  </field>
</mapping>
```

Custom converter 'instances' can be reused at the individual field level. In the example below, Dozer will invoke the custom converter to perform the field mapping.

```
<mapping>
  <class-a>org.dozer.vo.SimpleObj</class-a>
  <class-b>org.dozer.vo.SimpleObjPrime2</class-b>
  <field custom-converter-id="CustomConverterWithId">
    <a>field1</a>
    <b>field1Prime</b>
  </field>
</mapping>
```

CustomConverter instances need to be injected into the DozerBeanMapper if you need to do some manipulation with them before they are used in dozer. They can also be set programmatically on the DozerBeanMapper if you do not use Spring.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
  "http://www.springframework.org/dtd/spring-beans.dtd">

<beans default-lazy-init="false">

  <bean id="org.dozer.Mapper" class="org.dozer.DozerBeanMapper">
    <property name="mappingFiles">
      <list>
        <value>systempropertymapping1.xml</value>
        <value>dozerBeanMapping.xml</value>
        <value>injectedCustomConverter.xml</value>
      </list>
    </property>
    <property name="customConvertersWithId">
      <map>
        <entry key="CustomConverterWithId"
               ref="configurableConverterBeanInstance1"/>
        <entry key="CustomConverterWithId2"
               ref="configurableConverterBeanInstance2"/>
      </map>
    </property>
</bean>

</beans>
```

Sample custom converter implementation:

Note: Custom Converters get invoked when the source value is null, so you need to explicitly handle null values in your custom converter implementation.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
public class TestCustomConverter implements CustomConverter {

  public Object convert(Object destination, Object source,
      Class destClass, Class sourceClass) {
    if (source == null) {
      return null;
    }
    CustomDoubleObject dest = null;
    if (source instanceof Double) {
      // check to see if the object already exists
      if (destination == null) {
        dest = new CustomDoubleObject();
      } else {
        dest = (CustomDoubleObject) destination;
      }
      dest.setTheDouble(((Double) source).doubleValue());
      return dest;
    } else if (source instanceof CustomDoubleObject) {
      double sourceObj =
        ((CustomDoubleObject) source).getTheDouble();
      return new Double(sourceObj);
    } else {
      throw new MappingException("Converter TestCustomConverter "
          + "used incorrectly. Arguments passed in were:"
          + destination + " and " + source);
    }
  }
```

CustomConverters can also be injected into the DozerBeanMapper if you need to do some manipulation with them before they are used in dozer.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
  "http://www.springframework.org/dtd/spring-beans.dtd">

<beans default-lazy-init="false">

  <bean id="org.dozer.Mapper" class="org.dozer.DozerBeanMapper">
    <property name="mappingFiles">
      <list>
        <value>systempropertymapping1.xml</value>
        <value>dozerBeanMapping.xml</value>
        <value>injectedCustomConverter.xml</value>
      </list>
    </property>
    <property name="customConverters">
      <list>
         <ref bean="customConverterTest"/>
      </list>
    </property>
  </bean>

  <!-- custom converter -->
  <bean id="customConverterTest"
        class="org.dozer.converters.InjectedCustomConverter">
    <property name="injectedName">
      <value>injectedName</value>
    </property>
  </bean>

</beans>
```

19.1.1 Support for Array Types

You can specify a custom converter for Array types. For example, if you want to use a custom converter for mapping between an array of objects and a String you would use the following mapping notation. Dozer generically uses ClassLoader.loadClass() when parsing the mapping files. For arrays, java expects the class name in the following format.... [Lorg.dozer.vo.SimpleObj;

```
<converter
    type="org.dozer.converters.StringAppendCustomConverter" >
  <class-a>[Lorg.dozer.vo.SimpleObj;</class-a>
  <class-b>java.lang.String</class-b>
</converter>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

19.1.2 Support for primitives

You can specify a custom converter for primitive types. Just use the primitive wrapper class when defining the custom coverter mapping. In the following example, Dozer will use the specified custom converter when mapping between SomeObject and the int primitive type. Note that Dozer will also use the custom converter when mapping between SomeObject and the Integer wrapper type.

```
<converter type="somePackage.SomeCustomConverter" >
  <class-a>somePackage.SomeObject</class-a>
  <class-b>java.lang.Integer</class-b>
</converter>
```

19.1.3 Configurable Custom Converters

You can define a custom converter, which can be configured from mappings via configuration parameter. In this case you should implement ConfigurableCustomConverter interface instead of usual CustomConverter. Configurable converter has additional attribute provided in runtime - param. Parameter is provided using custom-converter-param attribute.

```
<mapping>
  <class-a>org.dozer.vo.BeanA</class-a>
  <class-b>org.dozer.vo.BeanB</class-b>
  <field custom-converter=
         "org.dozer.converters.MathOperationConverter"
         custom-converter-param="+">
    <a>amount</a>
    <b>amount</b>
  </field>
</mapping>
```

Configurable custom converter should be used when you have similar behaviour in many cases, which can be parametrized, but the number of conbinations is too high to do simple Custom Converter subclassing.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
public class MathOperationConverter
    implements ConfigurableCustomConverter {

  public Object convert(Object destinationFieldValue,
                        Object sourceFieldValue,
                        Class destinationClass,
                        Class sourceClass, String param) {
    Integer source = (Integer) sourceFieldValue;
    Integer destination = (Integer) destinationFieldValue;
    if ("+".equals(param)) {
      return destination.intValue + source.intValue();
    }
    if ("-".equals(param)) {
      return destination.intValue - source.intValue();
    }
  }

}
```

19.1.4 New Custom Converter API

While providing great deal of flexibility Custom Converter API described above is written on fairly low levele of abstraction. This results in converter, which code is difficult to understand and to reuse in other ways than plugging into Dozer mapping. However it is not uncommon situation when the same convertation logic should be called from a place other than bean mapping framework.

Latest version of Dozer gets shipped with new - cleaner API for defining custom converter, which gives you more obvious API while taking away certain part of control of the executions flow. The following example demonstrates simple, yet working converter using new API.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
public class NewDozerConverter
    extends DozerConverter<String, Boolean> {

  public NewDozerConverter() {
    super(String.class, Boolean.class);
  }

  public Boolean convertTo(String source, Boolean destination) {
    if ("yes".equals(source)) {
      return Boolean.TRUE;
    } else if ("no".equals(source)) {
      return Boolean.FALSE;
    }
    throw new IllegalStateException("Unknown value!");
  }

  public String convertFrom(Boolean source, String destination) {
    if (Boolean.TRUE.equals(source)) {
      return "yes";
    } else if (Boolean.FALSE.equals(source)) {
      return "no";
    }
    throw new IllegalStateException("Unknown value!");
  }

}
```

Note that Java 5 Generics are supported and you do not need to cast source object to desired type as previously.

19.1.5 Data Structure Conversions

There are cases where it is required to perform programmatic data structure conversion, say copy each odd element in a list as map key, but each even as map value. In this case it is needed to define transformation of the structure while relying on usual Dozer mapping support for individual values. For this purposes it is possible to use *MapperAware* interface, which injects current mapper instance inside custom converter.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
public static class Converter
    extends DozerConverter <List, Map> implements MapperAware {

  private Mapper mapper;

  public Converter() {
    super(List.class, Map.class);
  }

  public Map convertTo(List source, Map destination) {
    Map originalToMapped = new HashMap();
    for (Source item : source) {
      Target mappedItem = mapper.map(item, Target.class);
      originalToMapped.put(item, mappedItem);
    }
    return originalToMapped;
  }

<...>

  public void setMapper(Mapper mapper) {
    this.mapper = mapper;
  }

}
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

20 Custom Bean Factories

20.1 Custom Bean Factories

You can configure Dozer to use custom bean factories to create new instances of destination data objects during the mapping process. By default Dozer just creates a new instance of any destination objects using a default constructor. This is sufficient for most use cases, but if you need more flexibility you can specify your own bean factories to instantiate the data objects.

Your custom bean factory must implement the org.dozer.BeanFactory interface. By default the Dozer mapping engine will use the destination object class name for the bean id when calling the factory.

```
public interface BeanFactory {
  public Object createBean(Object source, Class sourceClass,
      String targetBeanId);
}
```

Next, in your Dozer mapping file(s) you just need to specify a bean-factory xml attribute for any mappings that you want to use a custom factory.

In the following example, the SampleCustomBeanFactory will be used to create any new instances of the InsideTestObjectPrime java bean data object.

```
<mapping>
  <class-a>com.example.vo.InsideTestObject</class-a>
  <class-b bean-factory=
      "com.example.factories.SomeCustomBeanFactory">
    com.example.vo.InsideTestObjectPrime
  </class-b>
</mapping>
```

If your factory looks up beans based on a different id than class name, you can specifiy a factory-bean-id xml attribute. At runtime the specified factory-bean-id will be passed to the factory instead of class name.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<mapping>
  <class-a>com.example.vo.InsideTestObject</class-a>
  <class-b bean-factory=
           "com.example.factories.SomeCustomBeanFactory"
           factory-bean-id="someBeanLookupId">
    com.example.vo.InsideTestObjectPrime
  </class-b>
</mapping>
```

20.1.1 Specifying Default Factories

Alternatively, bean factories can be specified in the default configuration section of any Dozer mapping file(s). The default factory would be used for any mappings in that file.

```
<configuration>
  <stop-on-errors>true</stop-on-errors>
  <wildcard>true</wildcard>
  <bean-factory>com.example.factories.SomeDefaultBeanFactory
  </bean-factory>
</configuration>
```

Bean factories can also be specified at the mapping level. The specified factory would be used for class-a and class-b.

```
<mapping bean-factory=
         "com.example.factories.SomeCustomBeanFactory">
  <class-a>com.example.vo.TestObject</class-a>
  <class-b>com.example.vo.TestObjectPrime</class-b>
</mapping>
```

20.1.2 Spring bean factory injection

Bean factories can be injected via Spring or similar inversion of control techniques.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<beans>
  <bean id="org.dozer.Mapper" class="org.dozer.DozerBeanMapper">
    <property name="mappingFiles">
      <list>
        <value>systempropertymapping1.xml</value>
        <value>dozerBeanMapping.xml</value>
      </list>
    </property>
    <property name="factories">
      <map>
        <!-- the key matches the name of the factory in the
             dozerBeanMapping.xml file -->
       <entry key="org.dozer.factories.SampleCustomBeanFactory">
         <ref bean="sampleCustomBeanFactory"/>
       </entry>
        <!-- more factories can be supplied with additional
             entry's -->
      </map>
    </property>
  </bean>

  <bean id="sampleCustomBeanFactory"
        class="org.dozer.factories.SampleCustomBeanFactory"/>
</beans>
```

By defining your factories as Spring beans you can then inject them into the DozerBeanMapper class.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

21 Custom Create Methods

21.1 Custom Create Methods

You can configure Dozer to use custom static create methods to create new instances of destination data objects during the mapping process. This can either be set at the field level or class level.

```
<mapping>
  <class-a create-method="someCreateMethod">
    org.dozer.vo.InsideTestObject
  </class-a>
  <class-b>org.dozer.vo.InsideTestObjectPrime</class-b>

  <field>
    <a>label</a>
    <b>labelPrime</b>
  </field>
</mapping>
```

Specifying a custom create method at the Field level....

```
<mapping>
  <class-a>org.dozer.vo.TestObject</class-a>
  <class-b>org.dozer.vo.TestObjectPrime</class-b>

  <field>
    <a>createMethodType</a>
    <b create-method="someCreateMethod">createMethodType</b>
  </field>
</mapping>
```

It is also possible to reference different class with static factory method. This is done by providing fully qualified type name and method name separated by dot.

```
...
<b create-method="org.dozer.factory.Factory.create">field</b>
...
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

22 Custom Get/Set Methods

22.1 Custom get() set() Methods

22.1.1 Mapping a field with no get() or set() methods

Use the attribute is-accessible to declare that the field can be accessed directly. Dozer is able to access private properties that do not have getter or setter methods.

```
<field>
  <a>fieldAccessible</a>
  <b is-accessible="true">fieldAccessible</b>
</field>
```

22.1.2 Custom Set() and Get() methods (bi-directional)

For those beans that might have unorthodox getter and setter methods, Dozer support user specified setter and getter methods. To make a bi-directional mapping in this case, look at the following example below. The source field in element A specifies a custom setter method and getter method using attributes.

```
<field>
  <a set-method="placeValue" get-method="buildValue">value</a>
  <b>value</b>
</field>
```

In this case we are mapping a String to an ArrayList by calling the addIntegerToList() method. Note that this is defined as a one-way field type since we can not map an ArrayList to a String.

```
<!-- we can not map a ArrayList to a String,
     hence the one-way mapping -->
<field type="one-way">
  <a>integerStr</a>
  <b set-method="addIntegerToList">integerList</b>
</field>
```

22.1.3 Overloaded Set() methods (bi-directional)

Sometimes set() methods can be overloaded. In order to chose the correct one you can add the class type as a parameter.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<field>
  <a>overloadGetField</a>
  <b set-method="setOverloadSetField(java.util.Date)">
    overloadSetField
  </b>
</field>
```

22.1.4 Iterate Method Mapping (bi-directional)

Dozer also supports iterate method level mapping. In the following example the List appleComputers will be iterated through and for each object the method addComptuer will be called. Any field that is denoted as type=iterate requires a hint. The get() method can return an Array, List, or Iterator.

```
<field>
  <a>appleComputers</a>
  <b set-method="addComputer" type="iterate">computers</b>
  <b-hint>org.dozer.vo.AppleComputer</b-hint>
</field>
```

Below is an example with iterate methods on both fields.

```
<field>
  <a set-method="addCar" get-method="myIterateCars"
     type="iterate">iterateCars</a>
  <b set-method="addIterateCar" type="iterate">iterateCars</b>
  <a-hint>org.dozer.vo.Car</a-hint>
  <b-hint>org.dozer.vo.Car</b-hint>
</field>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

23 Copy By Ref Mapping

23.1 Copying By Object Reference

Dozer supports copying an object by reference. No conversion/transformation is done for such objects. This approach allows to decrease a number of object allocations, but is applicable only when Java Beans are to be thrown away (Garbage Collected) after transformation. This approach is generally recommended for performance tuning of the mapping process when possible. Make sure that both object types are the same or you will run into casting problems. The default value is 'false'.

```
<field copy-by-reference="true">
  <a>copyByReference</a>
  <b>copyByReferencePrime</b>
</field>
```

This is also supported at the class level. Just define the classes you want to be copied by reference in the configuration block.

```
<configuration>
  ...
   <copy-by-references>
     <copy-by-reference>
       org.dozer.vo.NoExtendBaseObjectGlobalCopyByReference
     </copy-by-reference>
   </copy-by-references>
</configuration>
```

On the class level wildcard expressions are allowed. Copy by reference is applied via mask, which can inlcude multiple wildcard (*) characters.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<configuration>
  ...
   <copy-by-references>
     <copy-by-reference>
       org.dozer.vo.*
     </copy-by-reference>
     <copy-by-reference>
       org.dozer.*.vo.*DTO
     </copy-by-reference>
   </copy-by-references>
</configuration>
```

23.1.1 Referencing self (this) in a field mapping

Using a field mapping it is possible to map where N == 0 (self, or this). In the following example SimpleAccount is mapped to Address. It is also mapped to Account. Suppose Address was an attribute on Account. How could we map the values on SimpleAccount to that property? The answer is to use the keyword (this) to denote using the class being mapped as the source object.

```
<mapping>
  <classa>org.dozer.vo.self.SimpleAccount</classa>
  <classb>org.dozer.vo.self.Account</classb>
  <field>
    <a>this</a>
    <b>address</b>
  </field>
</mapping>

<mapping>
  <classa>org.dozer.vo.self.SimpleAccount</classa>
  <classb>org.dozer.vo.self.Address</classb>

  <field>
    <a>streetName</a>
    <b>street</b>
  </field>
</mapping>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

24 Inheritance Mapping

24.1 Inheritance Mapping

24.1.1 Reducing Mapping XML when using base class attributes

Properties that are of the same name do not need to be specified in the mapping xml file unless hints are needed.

If you are mapping subclasses that also have have base class attributes requiring mapping XML, you might be inclined to reproduce base class field maps in each subclass mapping element, like the following example:

```
<mapping>
  <class-a>org.dozer.vo.SubClass</class-a>
  <class-b>org.dozer.vo.SubClassPrime</class-b>

  <field>
    <!-- this is the same for all sub classes -->
    <a>superAttribute</a>
    <b>superAttr</b>
  </field>

  <field>
    <a>attribute2</a>
    <b>attributePrime2</b>
  </field>
</mapping>

<mapping>
 <class-a>org.dozer.vo.SubClass2</class-a>
 <class-b>org.dozer.vo.SubClassPrime2</class-b>

 <field>
   <!-- this is the same for all sub classes -->
   <a>superAttribute</a>
   <b>superAttr</b>
 </field>

 <field>
   <a>attribute2</a>
   <b>attributePrime2</b>
 </field>
</mapping>
```

In the previous mapping, some of the fields were from a common base class, but you had to reproduce them into each mapping of the sub classes.

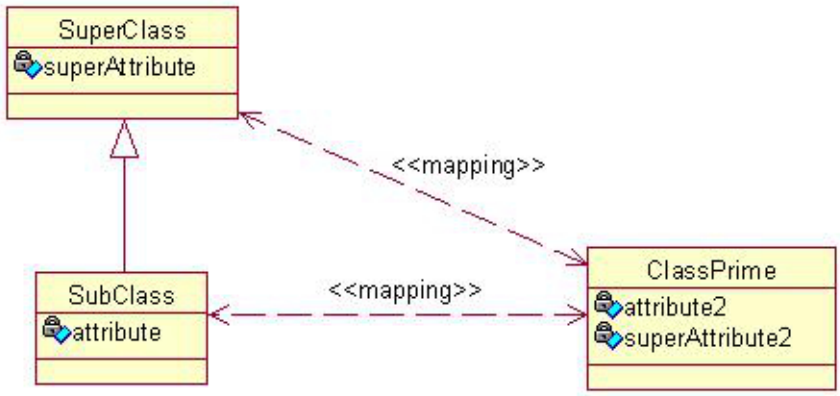
©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

However, a better way to do it would be to map the base class individually. This can be done for each base class (in the case of a larger heirarchy). Assuming the base class name, below is the refactored mapping xml:

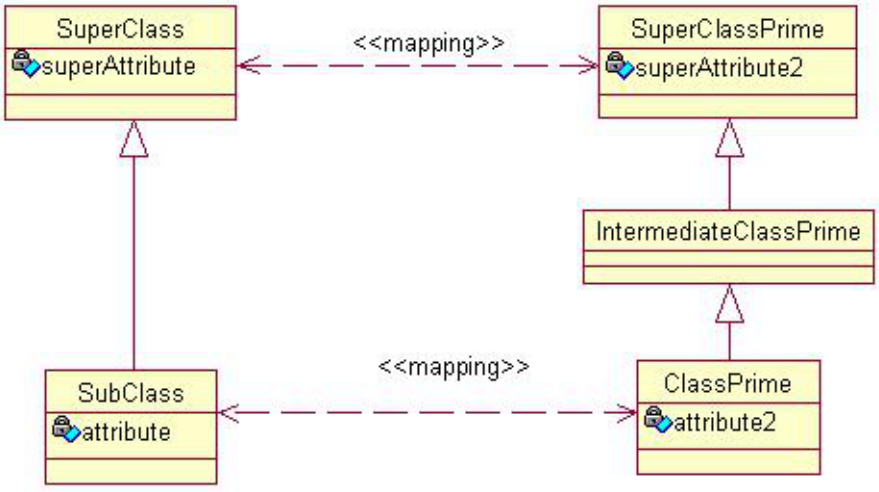
```
<mapping>
  <class-a>org.dozer.vo.SuperClass</class-a>
  <class-b>org.dozer.vo.SuperClassPrime</class-b>

  <field>
    <a>superAttribute</a>
    <b>superAttr</b>
  </field>
</mapping>

<mapping>
  <class-a>org.dozer.vo.SubClass</class-a>
  <class-b>org.dozer.vo.SubClassPrime</class-b>

  <field>
    <a>attribute</a>
    <b>attributePrime</b>
  </field>
</mapping>

<mapping>
  <class-a>org.dozer.vo.SubClass2</class-a>
  <class-b>org.dozer.vo.SubClassPrime2</class-b>

  <field>
    <a>attribute2</a>
    <b>attributePrime2</b>
  </field>
</mapping>
```

The following images explain some of the different scenarios dozer handles. Each diagram shows two mapped class hierarchies and existing relations, which Dozer recognizes and maps.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

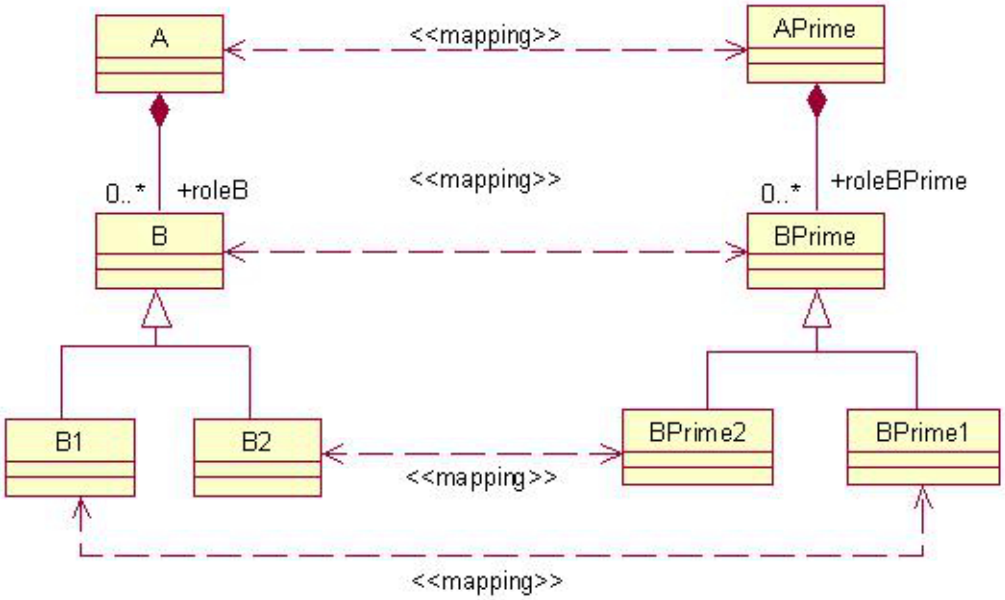


Scenario 1 shows that if you map SubClass to ClassPrime all attributes from SuperClass -> ClassPrime will be mapped as well.



Scenario 2 shows that Dozer has no limitations on the inheritance depth it analyzes to find parent mappings.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.



Scenario 3 shows that it is possible to map two collections with different subctypes of the same parent type. This is done by providing hints to the collection mapping, describing all potential subclasses.

```
<field>
  <a>aList</a>
  <b>bList</b>
  <a-hint>B1,B2</a-hint>
  <b-hint>BPrime1,BPrime2</b-hint>
</field>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

25 Dozer Events

25.1 Event Listening

By implementing the DozerEventListener interface dozer allows you to listen to 4 different events:

- mappingStarted
- mappingFinished
- preWritingDestinationValue
- postWritingDestinationValue

A DozerEvent object is passed into these callback methods which stores information about the ClassMap, FieldMap, Source object, destination object, and destination value. This will allow you to extend dozer and manipulate mapped objects at run-time. The interface is shown below:

```
public interface DozerEventListener {
  public void mappingStarted(DozerEvent event);
  public void preWritingDestinationValue(DozerEvent event);
  public void postWritingDestinationValue(DozerEvent event);
  public void mappingFinished(DozerEvent event);
}
```

The listeners that you create can be injected to the DozerBeanMapper using an IOC like Spring or set directly on your DozerBeanMapper instance by using the setEventListeners() method. Below is an example using Spring to inject an event listener:

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
 "http://www.springframework.org/dtd/spring-beans.dtd">

<beans default-lazy-init="false">

 <bean id="EventMapper" class="org.dozer.DozerBeanMapper">
  <property name="mappingFiles">
   <list>
    <value>dozerBeanMapping.xml</value>
   </list>
  </property>
  <property name="eventListeners">
   <list>
    <ref bean="eventTestListener"/>
   </list>
  </property>
 </bean>

 <bean id="eventTestListener"
       class="org.dozer.event.EventTestListener"/>

</beans>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

26 Expression Language

26.1 Expression Language

26.1.1 Usage

Dozer provides optional support for standard java expression language (javax.el).

Current support for expressions is start-up time only. Expressions are **not** resolved during each mapping, but rather during Xml mapping file loading procedure. Each attribute or node value can contain a valid EL expression ${}.

Dozer supports any EL implementation written against javax.el standard API. Functionality is tested with JUEL internally, but other EL providers should be working as well.

You can define global variables for the mapper in **variables** configuration block.

```
<configuration>
    <wildcard>true</wildcard>
    <variables>
        <variable name="type_name">org.dozer.sample.MyType
        </variable>
    </variables>
    <mapping>
        <class-a>${type_name}</class-a>
    ...
</configuration>
```

26.1.2 Enabling

EL support is an optional feature. If it is not enable it does not affect mapping performance neither it requires additional Jar dependencies to your project.

In order to enable EL expression execution *dozer.el.enabled* parameter should be set to true.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

27 Configuration

27.1 Configuration

Dozer configuration properties can be customized via an optional Dozer properties file. By default, Dozer will look for a file named dozer.properties to load configuration properties. If a properties file is not found or specified, default values will be used.

Dozer is distributed with an example dozer.properties file in /config that shows the various options. Just put the example file in your classpath and customize it.

An alternative Dozer properties file can be specified via the dozer.configuration system property. ex) -Ddozer.configuration=someDozerConfigurationFile.properties

Dozer Configuration Properties

Property Name	Description	Valid Values	Default
dozer.statistics.enabled	Specifies whether Dozer collects runtime mapping statistics. Note that statistics gathering imposes certain performance overhead. It is not recommended to enable this option in production settings under heavy load.	true|false	false
dozer.autoregister.jmx.bean	Specifies whether Dozer will auto register it's JMX beans with the PlatformMDBServer on startup.	true|false	true
dozer.el.enabled	Specifies whether during Xml mappings parsing Dozer will recognize EL expressions.	true|false	false
dozer.cache.converter.by.d	Specifies the max size for one of Dozers internal caches.	0 - Long.MAX_VALUE	10000
dozer.cache.super.type.ma	Specifies the max size for one of Dozers internal caches	0 - Long.MAX_VALUE	10000
org.dozer.util.DozerProxyR	Specifies implementation of DozerProxyResolver to be used	Valid class name	Default implementation
org.dozer.util.DozerClassLo	Specifies implementation of DozerClassLoader to be used	Valid class name	Default implementation

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

27.2 Debugging Initialization

One time Dozer initialization information can be optionally sent to System.out to help with debugging.

To enable this additional debugging info, set the dozer.debug system property.

ex) -Ddozer.debug=true

Types of information that will be sent to System.out are loading of configuration file, loading of custom xml mapping files, version info, classloader info

```
dozer:  Trying to find Dozer configuration file: dozer.properties
dozer:  Using URL [file:/local/subversion_projects/dozer/trunk/ta
rget/test-classes/dozer.properties] for Dozer global property con
figuration
dozer:  Reading Dozer properties from URL [file:/local/subversion
_projects/dozer/trunk/target/test-classes/dozer.properties]
dozer:  Finished configuring Dozer global properties
dozer:  Initializing Dozer.  Version: ${project.version}, Thread
Name:main
dozer:  Dozer JMX MBean [org.dozer.jmx:type=DozerStatisticsContr
oller] auto registered with the Platform MBean Server
dozer:  Dozer JMX MBean [org.dozer.jmx:type=DozerAdminController
] auto registered with the Platform MBean Server
dozer:  Initializing a new instance of the dozer bean mapper.
dozer:  Initializing a new instance of the dozer bean mapper.
dozer:  Using the following xml files to load custom mappings fo
r the bean mapper instance: [fieldAttributeMapping.xml]
dozer:  Trying to find xml mapping file: fieldAttributeMapping.x
ml
dozer:  Using URL [file:/local/subversion_projects/dozer/trunk/t
arget/test-classes/fieldAttributeMapping.xml] to load custom xml
 mappings
dozer:  Successfully loaded custom xml mappings from URL: [file:
/local/subversion_projects/dozer/trunk/target/test-classes/field
AttributeMapping.xml]
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

28 Logging

28.1 Logging

The logging facade framework in Dozer is SLF4J. It replaced Commons Logging, which was used in project previously until version 5.3.

Please refer to SLF4J Documentation for more information on setting up and configuring different logging implementations.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

29 Proxy Handling

29.1 Proxy Objects

29.1.1 Overview

Dozer supports mappings done on proxy objects. This is typically the case when using persistence framework, which supports sophisticated features like lazy-loading. In this case application is working with fake objects, containing the real objects encapsulated. Implementation of proxies is dependant on the technology you use. Generally speaking, there are two popular libraries for creating Java proxies (Cglib and Javassist). However, how particular framework makes uses of them could also vary. Dozer offers by default a generic way to handle simple proxy scenarios, both Javassist and Cglib. However it is strongly recommended to tune proxy handling behavior for your particular scenario.

29.1.2 Configuration

Proxy implementation is set-up by modifying configuration file. Currently, besides of default behavior, Hibernate and No-Proxy modes are supported. For the full list of the implementations, see the list of org.dozer.util.DozerProxyResolver interface implementations. The list could be retrieved from JavaDocs.

In case you do not map proxied objects - use NoProxy resolver, which imposes minimum performance overhead.

29.1.3 Custom Scenarios

For custom scenarios it is possible to provide your own implementation of org.dozer.util.DozerProxyResolver interface. It is configured in the same way as the standard classes.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

30 Statistics

30.1 Statistics

If you enable statistics, Dozer will collect a number of runtime mapping metrics. These statistics are global and can be accessed via the GlobalStatistics object. The statistics are also available via JMX. Dozer is distributed with a fully functional JMX DozerStatisticsControllerMBean.

Based on our profiling numbers, the overhead of enabling statistics is roughly 3-5%.

Dozer Statistics

Statistic Type	Description
Mapping Success Count	# of successful mappings
Mapping Failure Count	# of failed mappings
Mapping Overall Time	Overall time(ms) of successful mappings
Mapping Avg Time	Average time(ms) of successful mappings
Mapping Failure Exception Types	# of failures per exception type. This statistic shows what specific exceptions are being thrown when mappings fail
Mapping Failure Type	# of failures per source and destination class names. This statistic shows which specific mappings have failed per unique src class name and dest class name
Mapper Instances Count	# of DozerBeanMapper objects created. This should be a low number
Field Mapping Success Count	# of successful field mappings
Field Mapping Failure Count	# of failed field mappings
Field Mapping Failure Ignored Count	# of failed field mappings that were ignored during the mapping process. By default, Dozer throws an exception when any field mappings fail. But this behavior can be overridden. This statistic is useful to understand the impacts of setting stop-on-errors to false
Cache Hit Count	# of hits per internal Dozer cache type
Cache Miss Count	# of misses per internal Dozer cache type
Custom Converter Overall Time	Overall time(ms) of successful custom converter mappings
Custom Converter Success Count	# of successful custom converter mappings
Custom Converter Percentage of Overall Time	Percentage of overall time spent in custom converter mappings

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

31 JMX Integration

31.1 JMX Integration

Dozer can be managed via JMX. The Dozer distibution contains fully functional JMX MBeans. These MBeans can be found in the org.dozer.jmx package.

Dozer auto registers these JMX Beans with the PlatformMBeanServer. You can suppress this behavior with the following Dozer configuration property:

dozer.autoregister.jmx.beans = false

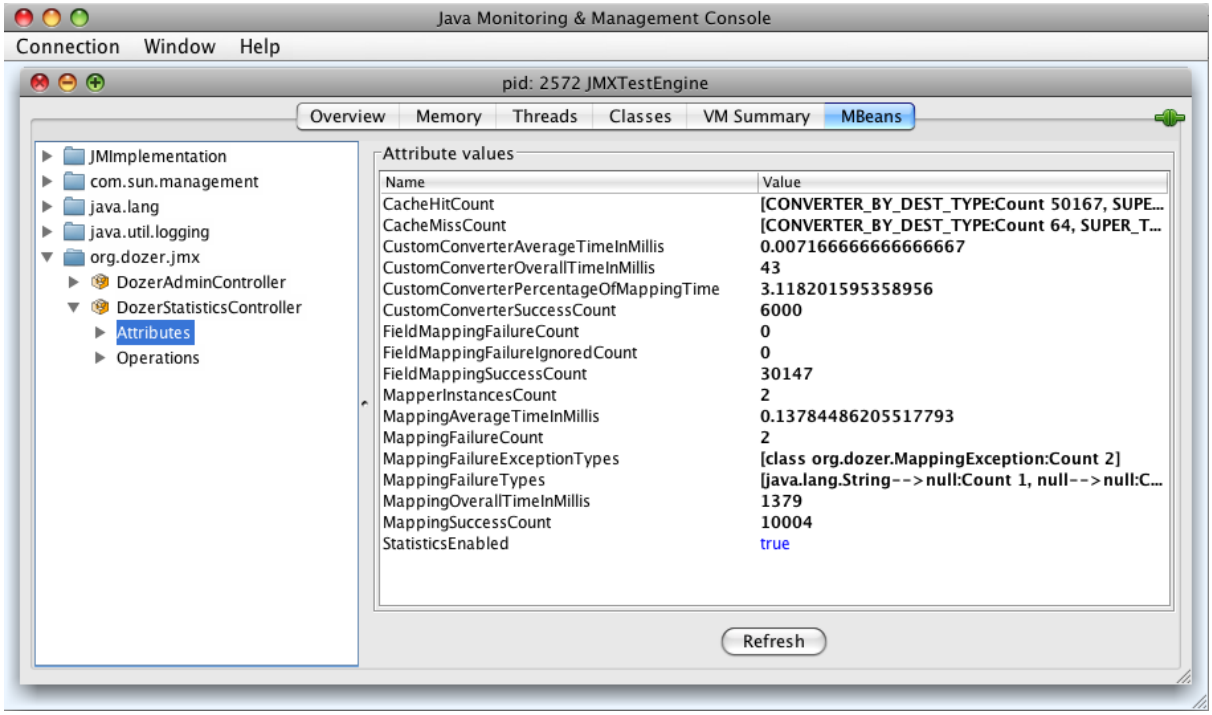
Dozer JMX MBeans

MBean	Description
DozerStatisticsControllerMBean	Runtime mapping statistics. The Statistics section contains more information on the types of statistics that are available.
DozerAdminControllerMBean	Admin functions such as enabling/disabling statistics gathering at runtime.

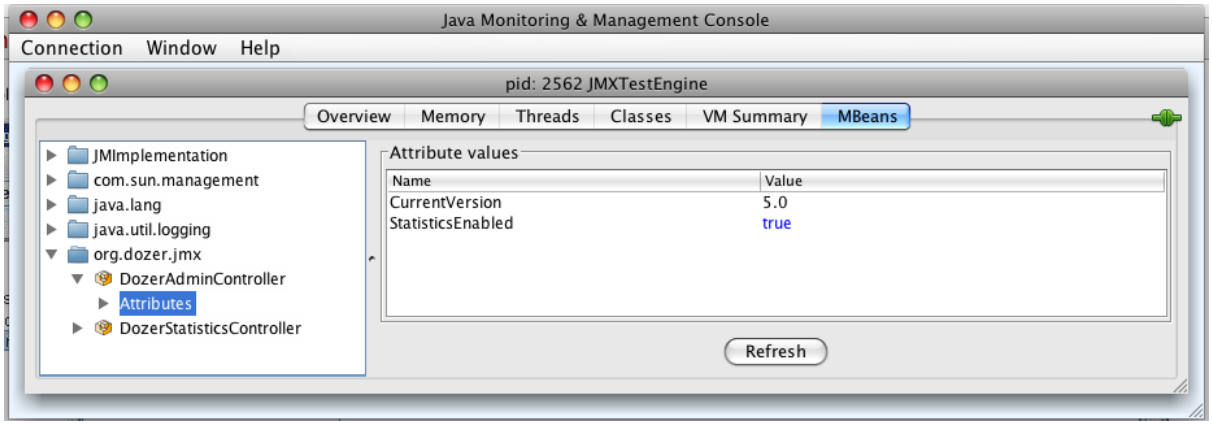
Note that JMX MBeans are potential source of memory leaks. MBeans should be disposed properly when application is stopped or restarted as in most of todays Web Containers there is no full JVM restart. The proper way to unregister all Dozer JMX Beans is to call destroy() method on DozerBeanMapper object.

31.1.1 Sample JMX Screen Shots

Dozer JMX MBeans via JConsole.....



©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.



©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

32 Spring Integration

32.1 Spring Framework Integration

Since version 5.5.0 Spring integration comes within additional module *dozer-spring*.

Add the DozerBeanMapperFactoryBean to your Spring configuration file. The mappingFiles property is where you should specify any custom dozer mapping files that you have created. This list can be empty if you don't have any custom mappings. It is also possible to set custom event listeners and bean factories.

Note that this Factory Bean supports Spring Resources, which means that you could load mapping Xml files by classpath mask for example.

```
<bean class="org.dozer.spring.DozerBeanMapperFactoryBean">
    <property name="mappingFiles"
              value="classpath*:/*mapping.xml"/>
    <property name="customConverters">
        <list>
            <bean class=
                  "org.dozer.converters.CustomConverter"/>
        </list>
    </property>
    <property name="eventListeners">
        <list>
            <bean class="org.dozer.listeners.EventListener"/>
        </list>
    </property>
    <property name="factories">
        <map>
            <entry key="id" value-ref="bean-factory-ref"/>
        </map>
    </property>
</bean>
```

IMPORTANT: You shoud define the Dozer mapper bean is defined as **singleton="true"**. You should configure the Mapper instance(s) this way so that you do not have to reload and reinitialize the mapping files for each individual mapping during the lifecycle of your app. Reinitializing the mapping files for each mapping would be inefficent and unnecessary. The DozerBeanMapper class is thread safe.

If you want to avoid clashing of mapping definitions for different functionality it is possible to define own DozerBeanMapper instance for each such use case.

A simpler way of registering mapper instance is without wrapping factory bean.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<bean id="org.dozer.Mapper" class="org.dozer.DozerBeanMapper">
  <property name="mappingFiles">
    <list>
      <value>dozer-global-configuration.xml</value>
      <value>dozer-bean-mappings.xml</value>
      <value>more-dozer-bean-mappings.xml</value>
    </list>
  </property>
</bean>
```

Using Spring to retrieve the Dozer Mapper.....

```
  Mapper mapper = yourSpringBeanFactory.getBean("mapperBeanName");
  DestinationObject destObject =
      mapper.map(sourceObject, DestinationObject.class);
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

33 JAXB and XMLBeans

33.1 3rd Party Object Factories

Dozer supports mapping of plain Java objects to frameworks that require instantiation of objects via certain convention of calling factory methods. To reproduce the expected behavior custom bean factories should be used.

33.1.1 Mapping JAXB Objects

Dozer has support for mapping POJOs to JAXB objects. Use the JAXBBeanFactory for any JAXB objects you want create.

```
<mapping>
  <class-a>org.dozer.vo.TestObject</class-a>
  <class-b bean-factory="org.dozer.factory.JAXBBeanFactory">
    org.dozer.vo.jaxb.employee.Employee
  </class-b>

  <field>
    <a>name</a>
    <b>firstName</b>
  </field>

  <field>
    <a>street</a>
    <b>address.street</b>
  </field>

</mapping>
```

33.1.2 Mapping XMLBeans

Dozer has support for mapping POJOs to XMLBeans objects. Use the XMLBeanFactory for any XMLBeans you want created. This factory will also be used for mapping any fields that need to be instantiated in a deep mapping that are not regular POJOs but are XMLBeans.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
<mapping wildcard="false">
  <class-a>org.dozer.vo.TestObject</class-a>
  <class-b bean-factory="org.dozer.factory.XMLBeanFactory">
    org.dozer.vo.GetWeatherByZipCodeDocument
  </class-b>

  <field>
    <a>one</a>
    <b>GetWeatherByZipCode.zipCode</b>
  </field>

</mapping>
```

The unit test:

```
// Map from TestObject to XMLBeans
TestObject to = new TestObject();
to.setOne("one");
GetWeatherByZipCodeDocument doc =
    mapper.map(to, GetWeatherByZipCodeDocument.class);
assertEquals(to.getOne(),
    doc.getGetWeatherByZipCode().getZipCode());

// Map from XMLBeans to TestObject
GetWeatherByZipCodeDocument res =
    GetWeatherByZipCodeDocument.Factory.newInstance();
GetWeatherByZipCode zipCode =
    res.addNewGetWeatherByZipCode();
zipCode.setZipCode("one");
TestObject to2 = mapper.map(res, TestObject.class);
assertEquals(res.getGetWeatherByZipCode().getZipCode(),
    to2.getOne());
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

34 Metadata Query Interface

34.1 Querying mapping metadata

This section will cover the mapping metadata interface. It provides easy to use read-only access to all important properties or aspects of the currently active mapping definitions.

The following sections will give some code examples how to use the mapping query interfaces. The most important interface in the whole process is org.dozer.metadata.MappingMetadata. An instance can be acquired by calling the getMappingMetadata() method of the DozerBeanMapper class. This call will initialize the mappings if the map method has not been called yet.

Consider the following mapping file:

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">

  <mapping>
    <class-a>org.dozer.vo.ClassA</class-a>
    <class-b>org.dozer.vo.ClassB</class-b>
    <field>
      <a>fieldA</a>
      <b>fieldB</b>
    </field>
  </mapping>

</mappings>
```

To begin querying the mapping definitions the following code is needed:

```
DozerBeanMapper mapper = new DozerBeanMapper();
mapper.setMappingFiles(listOfFiles);
MappingMetadata mapMetadata = beanMapper.getMappingMetadata();
```

Now that a reference to MappingMetadata is obtained we can start querying for a certain class mapping definition:

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
try {
  ClassMappingMetadata classMappingMetadata =
      mapMetadata.getClassMapping(ClassA.class, ClassB.class);
} catch (MetadataLookupException e) {
  // couldn't find it
}
```

When holding a reference to a ClassMappingMetadata interface, queries for individual field mappings can be executed:

```
try {
  FieldMappingMetadata fieldMetadata =
      classMetadata.getFieldMappingBySource("fieldA");
  // Destination: fieldB
  System.out.println("Destination: " +
      fieldMetadata.getDestinationName());
} catch (MetadataLookupException e) {
  // couldn't find it
}
```

For extensive documentation on the different interfaces please refer to the JavaDoc.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

35 FAQ

35.1 Frequently Asked Questions

35.1.1 Common

- What types of data objects are supported?
- Will Dozer automatically perform data type conversions?
- Does Dozer automatically map fields with matching property names?
- Is Dozer recursive?
- Will the getter and setter methods be invoked when fields are mapped?
- Are Collections and Arrays supported?
- Are Map type objects(i.e HashMap) supported?
- Are abstract classes, inheritance, and interface mapping supported?
- Can Dozer be configured via Spring?
- Which types of data mappings do I need a custom xml mapping definition for?
- If my src and dest object have all matching attribute names, do I need to specify any xml mapping definitions at all?
- For mappings that require an xml mapping definition, is the mapping definition bi-directional, or do I need 2 xml definitions if I need to map the two objects both ways?
- How are the custom xml mapping files loaded?
- Can I load a mapping file that is not in the classpath?
- How can I tell if Dozer is initializing correctly and loading my xml mapping files?
- How does Dozer perform?
- Which JDK versions are supported?
- Is Dozer in the maven repository?
- Is Dozer good for the environment?

35.1.2 Advanced

- Can I implement custom mapping logic between 2 data types and have Dozer invoke this custom logic when it's performing mappings?
- Can I map one field into another field that is nested n layers deep in the destination object?
- How do I map multiple fields to a single field?
- If I am mapping data objects that have bi-directional relationships, will it result in an infinite loop and eventual stack overflow error?
- How do I map an object contained in a collection to a field?
- How do I map a Complex object to a HashMap and vice versa?
- How do I map fields that don't have corresponding getter/setter methods?
- Some of my data objects don't have public constructors. Does Dozer support this use case?
- Does Dozer support JDK 1.5 enums?
- Does Dozer support XMLBeans and JAXB generated data objects?
- Is there an Eclipse plugin or visual editor for Dozer?
- When mapping collections, how do I tell Dozer what type of data objects I want in the destination collection?
- How can I tell Dozer to bypass mapping null or empty string values?

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

- How do I enable Dozer to collect runtime mapping statistics?

35.1.3 Tips, Tricks, and Suggestions

- Should I encapsulate logic that copies data between objects?
- Should I write unit tests for data mapping logic that I use Dozer to perform?
- Should the Dozer mapper be configured as a Singleton?
- Is it better to have 1 large xml mapping file or to have multiple smaller mapping files?
- What is the best way to view runtime mapping statistics?
- What are the best ways to debug Dozer?
- What is the best way to setup the global configuration?
- What is the best way to submit a bug, feature request, or patch?

35.2 Answers

35.2.1 What types of data objects are supported?

Dozer uses reflection to access data object properties, so it is designed to work with data objects that have corresponding getter and setter methods for its fields. For example, a data object that has a field named "message" should have getMessage and setMessage methods. Data objects that don't follow this pattern are also supported, but will most likely require a custom mapping definition. For these unorthodox data objects, you can tell Dozer to directly access fields(including private) and/or explicitly specify which get/set methods to use.

35.2.2 Will Dozer automatically perform data type conversions?

Yes. Most scenarios are supported out of the box. These include primitives, Java Wrapper Objects, Number subclasses, Dates, Calendar, Collections, Arrays, Maps, and Complex types

35.2.3 Does Dozer automatically map fields with matching property names?

Yes. All fields with matching property names are implicitly mapped. It would be atypical usage, but you could suppress this behavior by setting wilcard="false".

35.2.4 Is Dozer recursive?

Yes. Dozer recursively maps the entire object graph for all fields.

35.2.5 Will the getter and setter methods be invoked when fields are mapped?

Yes. You can bypass this default behavior by explicitly specifying is-accessible="true" for any of your mappings. If is-accessible is specified, the field(including private fields) is accessed directly and the getter/setter methods are bypassed. It is not recommended that you set is-accessible="true", unless you are dealing with an unorthodox data object that does not contain any getter or setter methods.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

35.2.6 Are Collections and Arrays supported?

Yes. Dozer automatically maps between collection types and automatically performs any type conversion.

35.2.7 Are Map type objects(i.e HashMap) supported?

Yes. All Java Map data types are supported in addition to any Custom map data types.

35.2.8 Are abstract classes, inheritance, and interface mapping supported?

Yes.

35.2.9 Can Dozer be configured via Spring?

Yes. Refer to Spring Integration section of the documentation.

35.2.10 Which types of data mappings do I need a custom xml mapping definition for?

Only fields that can't be implicitly mapped by matching on field name, need a custom xml mapping definition. Ideally, the vast majority of your field mappings can be performed automatically and only the few exceptional cases will need an explicit field mapping in the xml mapping file.

35.2.11 If my src and dest object have all matching attribute names, do I need to specify any xml mapping definitions at all?

Nope. Just invoke the mapper. You don't need any explicit xml mapping entries for this combination of source and destination object.

35.2.12 For mappings that require an xml mapping definition, is the mapping definition bi-directional, or do I need 2 xml definitions if I need to map the two objects both ways?

All mapping definitions are bi-directional, so you only need one mapping definition. You can map a --> b and b--> a using this single mapping definition.

35.2.13 How are the custom xml mapping files loaded?

Dozer will search the entire classpath looking for the specified file(s).

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

35.2.14 Can I load a mapping file that is not in the classpath?

Yes, you can load files from outside the classpath by prepending "file:" to the resource name. Ex) "file:c:\somedozermapping.xml"

35.2.15 How can I tell if Dozer is initializing correctly and loading my xml mapping files?

Set the -Ddozer.debug system property. If this is set, Dozer initialization information is also sent to System.out. If you are familiar with log4j, this is similar to the -Dlog4j.debug system property

35.2.16 How does Dozer perform?

We believe Dozer performs very well and performance is a high priority for us. We have spent a significant amount of time profiling the code and optimizing bottlenecks.

Performance is going to depend on the complexity of the use case and the number of fields mapped. In our performance tests for "average" mapping scenarios, the class mapping times vary from 1/8 of a millisecond to 2 milliseconds. This roughly equates to 50 - 450 field mappings per millisecond. However, the number of variables in any decent benchmark makes it almost impossible to transfer these results into reasonable conclusions about the performance of your own application. Your application is different and you will have unique use cases.

Dozer has been successfully implemented on large, very high transactional enterprise systems, without any resulting performance issues. But we always recommend that you run performance tests on your application to determine the actual performance costs within your system. You can decide for yourself whether those costs are acceptable in the context of the entire system.

If you want to see actual Dozer runtime mapping statistics within the context of a system/application, you can enable Dozer statistics. This would be a good way to determine actual mapping times as a percentage of overall system performance. The best way to view the stats is via the Dozer JMX Beans. With the 3.2 release, these JMXBeans are auto registered with the platform mdb server. JConsole is a good way to easily view the MBeans.

Based on our profiling numbers, the overhead of enabling statistics is roughly 3-5%.

35.2.17 Which JDK versions are supported?

JDK 1.5 and above.

35.2.18 Is Dozer in the maven repository?

Yes and we will continue to do our best to get future releases of Dozer uploaded into the repository.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
    <dependency>
      <groupId>net.sf.dozer</groupId>
      <artifactId>dozer</artifactId>
      <version>${project.version}</version>
    </dependency>
```

35.2.19 Is Dozer good for the environment?

Yes, dozer does not burn any fossil fuels and is within the EPA's recommended emissions.

35.2.20 Can I implement custom mapping logic between 2 data types and have Dozer invoke this custom logic when it's performing mappings?

Yes. A very useful feature provided by Dozer is the concept of custom converters. Custom converters are used to perform custom mapping between two objects. In the Configuration block, you can add some XML to tell Dozer to use a custom converter for certain class A and class B types. When a custom converter is specified for a class A and class B combination, Dozer will invoke the custom converter to perform the data mapping instead of the standard mapping logic.

```
<custom-converters>
  <converter type="org.dozer.converters.SomeCustomConverter" >
    <class-a>org.dozer.vo.SomeCustomDoubleObject</class-a>
    <class-b>java.lang.Double</class-b>
  </converter>
</custom-converters>
```

35.2.21 Can I map one field into another field that is nested n layers deep in the destination object?

Yes. Dozer supports dot notation for nested fields. As with other dozer field mappings, these are bi-directional.

```
<field>
  <a>someNestedObj.someOtherNestedObj.someField</a>
  <b>someOtherField</b>
</field>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

35.2.22 How do I map multiple fields to a single field?

Dozer doesn't currently support this. And because of the complexities around implementing it, this feature is not currently on the road map. A possible solution would be to wrap the multiple fields in a custom complex type and then define a custom converter for mapping between the complex type and the single field. This way, you could handle the custom logic required to map the three fields into the single one within the custom converter.

35.2.23 If I am mapping data objects that contain bi-directional relationships, will it result in an infinite loop and eventual stack overflow error?

No. Dozer has built in logic that prevents infinite loops for bi-directional data object relationships

35.2.24 How do I map an object contained in a collection to a field?

You would use indexed based mapping.

```
<field>
  <a>usernames[0]</a>
  <b>username1</b>
</field>
```

35.2.25 How do I map a Complex object to a HashMap and vice versa?

You can map entire complex objects directly to a java.util.Map and vice versa. When doing this you need to explicitly define a unique map-id for the mapping. This is used when determining which map to use at run-time. Every attribute on the complex type will be mapped to the java.util.Map. You DO NOT need to explicitly define these mappings. If the attribute name is not the same as the map key just set the key attribute for a custom field mapping.

```
<mapping map-id="myTestMapping">
  <class-a>org.dozer.vo.map.SomeComplexType</class-a>
  <class-b>java.util.Map</class-b>
  <field>
    <a>stringProperty2</a>
    <b key="myStringProperty">this</b>
  </field>
```

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

35.2.26 How do I map fields that don't have corresponding getter/setter methods?

You can tell Dozer to directly access fields(including private fields) by specifying is-accessible="true"

```
<field>
  <a>fieldA</a>
  <b is-accessible="true">fieldB</b>
</field>
```

35.2.27 Some of my data objects don't have public constructors. Does Dozer support this use case?

Yes. When creating a new instance of the destination object if a public no-arg constructor is not found, Dozer will auto detect a private constructor and use that. If the data object does not have a private constructor, you can specify a custom BeanFactory for creating new instances of the destination object.

35.2.28 Does Dozer support JDK 1.5 enums?

Yes. Enum to Enum mapping is automatically handled.

35.2.29 Does Dozer support XMLBeans and JAXB generated data objects?

Dozer supports mapping POJOs to XMLBeans objects. Use the XMLBeanFactory for any XMLBeans you want created. This factory will also be used for mapping any fields that need to be instantiated in a deep mapping that are not regular POJOs but are XMLBeans.

Dozer has support for mapping POJOs to JAXB objects. Use the JAXBBeanFactory for any JAXB objects you want created.

35.2.30 Is there an Eclipse plugin or visual editor for Dozer?

No, but we think it would be a great addition. It would be very powerful to be able to graphically map 2 objects and have the custom xml definitions auto generated, along with being able to visually view a mapping definition. If anyone has expertise in creating eclipse plugins and is interested on working on this feature, please let us know!

35.2.31 When mapping collections, how do I tell Dozer what type of data objects I want in the destination collection?

Hints are supported to handle this use case. Hints are not required if you are using JDK 1.5 Generics because the types can be auto detected by Dozer. But if you are not using generics, to convert a Collection/Array to a Collection/Array with different type objects you can specify a Hint to let Dozer know what type of objects you want created in the destination list. If a Hint is not specified for the

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

destination field, then the destination Collection will be populated with objects that are the same type as the elements in the src Collection.

```
<field>
  <a>someList</a>
  <b>otherList</b>
  <b-hint>org.dozer.vo.TheFirstSubClassPrime</b-hint>
</field>
```

35.2.32 How can I tell Dozer to bypass mapping null or empty string values?

You can bypass the mapping of null values by specifying map-null="false". If this is specified, the dest field mapping is bypassed at runtime and the destination value setter method will not be called if the src value is null. This can be specified at the mapping or class level.

You can bypass the mapping of empty String values by specifying map-empty-string="false". If this is specified, the dest field mapping is bypassed at runtime and the destination value setter method will not be called if the src value is an empty String. This can be specified at the mapping or class level

35.2.33 How do I enable Dozer to collect runtime mapping statistics?

In your dozer.properties file set "dozer.statistics.enabled=true"

35.2.34 Should I encapsulate logic that copies data between objects?

It is our opinion that you should. Regardless of whether you use Dozer to perform data mapping between objects, we believe this is a good design pattern that promotes reuse, encapsulates the underlying implementation, and makes the code unit testable in isolation. These "Assembler" interfaces encapsulate the logic that is responsible for taking a src object and mapping the data into a dest object. Using assembler type of classes gives you the flexibility of being able to modify the underlying mapping implementation without impacting clients or the contract. One other important benefit of using Assemblers is that it makes writing unit tests specific for the mapping a lot easier and more focused. If you ever need to determine if a particular bug is due to mapping of objects, it is simple to write an Assembler unit test that reproduces the use case. If you encapsulate your data mapping logic, you could use Dozer for most of mappings and if you have a real corner case, you have the flexibility to hand code mappings for any objects or fields. For example, you could run your mapping through Dozer to map 99% of your fields and then have a manual mapping for some odd ball field. This would happen all within the Assembler without the client having any knowledge of the underlying implementation.

It seems to work best if these assembler type of classes are "dumb" and are only responsible for simply copying data from the source object into the destination object. Any complex postprocessing business logic that needs to be performed on the destination object can be done at a higher level in classses that have more responsibility.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

The following is a simple example of an assembler type class that uses Dozer for its underlying implementation.

```
public class SomeAssemblerImpl implements SomeAssembler {

  private Mapper dozerMapper;

  public DestObject assembleDestObject(SrcObject src) {
    return dozerMapper.map(src, DestObject.class);
  }

}
```

35.2.35 Should I write unit tests for data mapping logic that I use Dozer to perform?

Absolutely. And of course, we strongly recommend writing the unit test(s) first. Even if you don't use Dozer to perform the data mapping between two objects, this logic still needs isolated unit tests. Data mapping logic(especially hand coded) is error prone and having a unit test is invaluable. Typically mapping between two objects is required in multiple areas of a system, so a focused unit test of the central mapping logic enables you to test the data mapping logic in isolation. The great thing about encapsulating data mapping logic and having unit tests for the logic is that you can easily switch out the underlying implementation.

For existing systems that are wanting to migrate to Dozer, we recommend first encapsulating any existing hand coded data mapping into an assembler type of class and write unit tests for it. Then switch out the hand coded mapping logic with Dozer and the unit tests will be your safety net. The migration to Dozer can be incremental and this is probably the best strategy for exisiting systems.

Regardless of whether or not you use Dozer, unit testing data mapping logic is tedious and a necessary evil, but there is a trick that may help. If you have an assembler that supports mapping 2 objects bi-directionally, in your unit test you can do something similar to the following example. This also assumes you have done a good job of implementing the equals() method for your data objects. The idea is that if you map a source object to a destination object and then back again, the original src object should equal the object returned from the last mapping if fields were mapped correctly. In the test case, you should populate all the possible fields in the original source object to ensure that all of the fields are accounted for in the mapping logic.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
public void testAssembleSomeObject() throws Exception {
  SrcObject src = new SrcObject();
  src.setSomeField("somevalue");
  src.setSomeOtherField("make sure you set all the src fields "
    + "with values so that you fully test the data mappings");

  DestObject dest = assembler.assembleDestObject(src);
  SrcObject mappedSrc = assermbler.assembleSrcObject(dest);

  assertEquals("fields not mapped correctly", src, mappedSrc);
}
```

It is also good practice to verify that your assembler handles null values properly. In the following test case none of the source fields are populated. If the assembler doesn't properly handle null values, an exception will be thrown when the assembler is invoked.

```
public void testAssembleSomeObject_NullValues() throws Exception {
  SrcObject src = new SrcObject();

  DestObject dest = assembler.assembleDestObject(src);
  SrcObject mappedSrc = assermbler.assembleSrcObject(dest);

  assertEquals("fields not mapped correctly", src, mappedSrc);
}
```

35.2.36 Should the Dozer mapper be configured as a Singleton?

Yes. Mapper instance(s) should be setup as a Singleton. For every instance of the DozerBeanMapper, the mapping files are loaded and parsed. You should configure the Mapper as a singleton so that you only incur the cost of loading and initializing the mapping files 1 time. The DozerBeanMapper class is thread safe.

35.2.37 Is it better to have 1 large xml mapping file or to have multiple smaller mapping files?

We recommend componentizing your mapping files instead of having 1 large mapping file.

35.2.38 What is the best way to view runtime mapping statistics?

The best way to view the runtime stats is via the Dozer JMX Beans. These JMXBeans are auto registered with the platform mdb server at startup. JConsole is a good way to easily view the MBeans.

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

35.2.39 What are the best ways to debug Dozer?

You can specify the -Ddozer.debug system property to view the one time initialization information. You will see output similar to the following....

```
dozer: Trying to find Dozer configuration file: dozer.properties
dozer: Using URL [file:/local/subversion_projects/dozer/trunk/ta
rget/test-classes/dozer.properties] for Dozer
global property configuration
dozer: Reading Dozer properties from URL
[file:/local/subversion_projects/dozer/trunk/target/test-classes
/dozer.properties]
dozer: Finished configuring Dozer global properties
dozer: Initializing Dozer. Version: ${project.version}, Thread N
ame:main
dozer: Dozer JMX MBean [org.dozer.jmx:type=DozerStatisticsContro
ller] auto registered with the Platform MBean
Server
dozer: Dozer JMX MBean [org.dozer.jmx:type=DozerAdminController]
 auto registered with the Platform MBean
Server
dozer: Initializing a new instance of the dozer bean mapper.
dozer: Initializing a new instance of the dozer bean mapper.
dozer: Using the following xml files to load custom mappings for
 the bean mapper instance:
[fieldAttributeMapping.xml]
dozer: Trying to find xml mapping file: fieldAttributeMapping.xm
l
dozer: Using URL [file:/local/subversion_projects/dozer/trunk/ta
rget/test-classes/fieldAttributeMapping.xml]
to load custom xml mappings
dozer: Successfully loaded custom xml mappings from URL:
[file:/local/subversion_projects/dozer/trunk/target/test-classes
/fieldAttributeMapping.xml]
```

To debug individual field mappings between classes, set the logging level "org.dozer.MappingProcessor=DEBUG". For example, if you are using log4j you would add the following entry to your log4j configuration file "log4j.category.org.dozer.MappingProcessor=DEBUG". This will show you every field mapping that Dozer performs along with the actual source and destination values. You will see output similar to the following....

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.

```
MAPPED: SimpleObj.field1 --> SimpleObjPrime.field1 VALUES:
one --> one MAPID: someMapId
MAPPED: SimpleObj.field2 --> SimpleObjPrime.field2 VALUES:
2 --> 2 MAPID: someMapId
MAPPED: SimpleObj.field3 --> SimpleObjPrime.field3 VALUES:
3 --> 3 MAPID: someMapId
MAPPED: SimpleObj.field4 --> SimpleObjPrime.field4 VALUES:
44.44 --> 44.44 MAPID: someMapId
MAPPED: SimpleObj.field6 --> SimpleObjPrime.field6 VALUES:
66 --> 66 MAPID: someMapId
```

35.2.40 What is the best way to setup the global configuration?

We recommend having a separate mapping xml file for global configuration. You could name it something similar to dozer-global-configuration.xml. Sample global configuration file.....

```
<?xml version="1.0" encoding="UTF-8"?>
<mappings xmlns="http://dozer.sourceforge.net"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://dozer.sourceforge.net
          http://dozer.sourceforge.net/schema/beanmapping.xsd">

  <configuration>
    <stop-on-errors>true</stop-on-errors>
    <date-format>MM/dd/yyyy HH:mm</date-format>
    <wildcard>false</wildcard>
    <custom-converters>
      <converter type="org.dozer.converters.TestCustomConverter" >
        <class-a>org.dozer.vo.CustomDoubleObject</class-a>
        <class-b>java.lang.Double</class-b>
      </converter>
     </custom-converters>
  </configuration>

</mappings>}
```

35.2.41 What is the best way to submit a bug, feature request, or patch?

We value your suggestions and appreciate everyone that takes the time to submit a support request. Please submit all requests via Dozer's Sourceforge project page

©2014, Franz Garsombke, Matt Tierney, Dmitry Buzdin • ALL RIGHTS RESERVED.