Table of Contents

News

About

User Guide

Schema

Eclipse Plugin

Community

Release Notes

Dozer

What is Dozer?

Dozer is a Java Bean to Java Bean mapper that recursively copies data from one object to another. Typically, these Java Beans will be of different complex types.

Dozer supports simple property mapping, complex type mapping, bi-directional mapping, implicit-explicit mapping, as well as recursive mapping. This includes mapping collection attributes that also need mapping at the element level.

Please read the about page for detailed information on Dozer.

News

2017-04-19 6.0.0 is coming!

Soon!

Archived News

See for pre-2017 news.

Archived News - Pre-2017

Dozer 5.5.1 is out *2014-04-22*

Release 5.5.1 is available for download and should appear in Maven Central shortly. Project has migrated to bintray for hosting binary releases and as an alternative repository location. Check release notes for the list of issues closed.

Dozer 5.4.0 is out *2012-12-01*

Release 5.4.0 is available for download. This is the first release after moving project to GitHub and already few new contributors have joined.

For discussions about new features post here Google Group. For technical questions use StackOverflow All issue tracking will be done here GitHub Old SourceForge Forums and Issue tracker will not be active anymore. They will be disabled once migration is finished.

Dozer is on GitHub *2012-06-28*

After some inactivity period Dozer has finally moved to GitHub. Source code will be maintained there from now on. This transition is a huge step forward for the project as the barrier for contribution is significantly lower on GitHub compared to SourceForge. It is really easy to create a Pull request and merge it afterwards. Transition of the issue trackers and support forums will be performed as well. We are looking forward to accept contributions from developers in forms of patches and help on migration to GitHub. Write to buzdin at gmail if you are interested.

Release 5.4.0 is currently in development and will be released this summer.

Release 5.3.2 Now Available *2011-02-15*

Release 5.3.2 is available for download. This is maintenance release, which finalizes internal builder API implementation and delivers several bugfixes. Experimental annotations support for simple mappings has been added.

Additionally, thanks to Lenkwe Makhubela, Dozer Eclipse plugin is finally working on Eclipse 3.6.

Refer to release notes for detailed information.

Release 5.3.1 Now Available *2010-11-15*

Release 5.3.1 is available for download. This is maintenance release bringing compatibility with Apache Camel framework and fixing a few bugs found in 5.3.0. Upgrade is highly recommended for 5.3.0 users.

Release 5.3.0 Now Available *2010-10-10*

Release 5.3.0 is available for download. This release brings several major changes to Dozer. First is migration to SLF4J logging framework. Second is introduction of alternative way of providing custom mappings via Java API. In the meanwhile we reached 50000 downloads from the website. This is excluding Maven repository.

Refer to release notes for detailed information.

Release 5.2.2 Now Available *2010-06-09*

Release 5.2.2 is available for download. This is a maintenance release fixing several issues. Upgrade is recommended for all 5.x.x users.

Thanks to everyone contributing for this release!

Please check release notes for more information.

Release 5.2.1 Now Available *2010-04-28*

Release 5.2.1 is available for download. Important new features are Dozer OSGi bundle and Expression Language support. Several bugs has been fixed as well. Upgrade is recommended for all 5.x.x users.

Thanks to everyone contributing for this release!

Please check release notes for more information.

Release 5.2.0 Now Available *2010-01-24*

Dozer team is happy to announce release 5.2.0. This version fixes fourteen bugs of different kind and provides several new features, such as referencing current mapper object inside a custom converter. Upgrade is recommended for all 5.x.x users.

Thanks to everyone contributing for this release!

Please check release notes for more information.

Release 5.1 Now Available *2009-08-25*

This is a maintenance release addressing more than a dozen bugs. Upgrade is recommended for everyone using 5.0 version. Several nasty bugs in area of processing complex type hierarchies have been fixed. The most valuable addition to this version is a new custom converter API, which supports Generics.

Thanks to everyone contributing patches and ideas for this release!

Please check release notes for more information

Dozer Eclipse Plugin 0.8.0 Now Available *2009-06-24*

This is the first release with a fully featured GUI around the Mapping XML.

Features are:

- GUI for the mapping XML. Based on Eclipse Forms.
- Content Assist and XML Validation for the XML View.
- Compatible with Dozer 4.x and 5.x

Thanks to Angelo Zerr who wrote the DOM/SSE Databinding that the Plugin uses.

Please check here for more information.
installation
usage

Release 5.0 Now Available *2009-03-03*

This release contains changes that are not backwards compatible with prior versions of Dozer. There is a 5.0 migration guide in the release notes section, which outlines these changes.

Key 5.0 changes:

- Added Generics to public api
- Switched to XSD instead of DTD
- Simpler packaging
- Upgraded 3rd party dependencies
- Upgraded code to use jdk 1.5 features
- Various feature requests and bug fixes

Thanks to everyone that helped out with this release and submitted tickets. Please let us know if you find any issues with the release and we will do our best to fix them soon as possible.

Please check here for more information and the 5.0 migration guide. release notes

Release 4.4.1 Now Available *2009-01-31*

This release fixes classloading issues for custom ClassLoaders. These classloading issues were introduced in the 4.3 release.

Please check here for more release notes

Release 4.4 Now Available *2008-12-27*

This release contains various bug fixes and feature requests.

Thanks to everyone that submitted bugs and contributed to the release.

Please check here for more release notes

Release 4.3 Now Available *2008-12-03*

This is a maintenance release targeting mostly bug fixes. Note that this release changes classloading behaviour. If you are experiencing classloading problems in any application container, please tell us urgently.

Thanks to our contributors for submitted bug reports and patches, which helped greatly during this release development.

Contributors to the project are more than welcome as usual!

Please check release notes for detailed information.

Release 4.2.1 Now Available *2008-06-22*

This minor release contains a fix for the stop-on-error bug.

Please check here for more release notes

Release 4.2 Now Available *2007-12-16*

This release contains various bug fixes.

Thanks to everyone that submitted bugs and contributed to the release.

Please check here for more release notes

Release 4.1 Now Available *2007-09-22*

This release contains bug fixes and feature requests, along with internal refactoring.

Thanks again to everyone that submitted bug and feature requests, along with everyone that contributed to the release.

Please check here for more release notes

Release 4.0 Now Available *2007-07-15*

This release contains bug fixes and feature requests, along with internal refactoring. Some new features include indexed mapping within deep mapping, improved support of Map backed properties, and configuration of auto string trimming.

The most significant refactoring was around Map backed properties. The refactoring did not change the public Dozer API or the xml dtd, but if you are currently using the Map backed property features of Dozer please thoroughly regression test your system after upgrading to 4.0 to verify that any expected behavior remains unchanged for your specific use cases. We do have a good amount of unit tests

around the Map backed property feature, but please let us know if you experience any issues after upgrading and we will do everything we can to resolve it soon as possible.

Thanks again to everyone that submitted bug and feature requests, along with everyone that contributed to the release.

Please check here for more release notes

Dozer hits 10,000 Downloads *2007-05-19*

We released dozer around 2 years ago and never expected this many downloads for such a niche framework. Thanks to the community for contributing and using our tool. Let's hope for 10,000 more downloads!

Release 3.4 Now Available *2007-05-19*

This release contains bug fixes and feature requests.

Please check here for more release notes

Release 3.3.1 Now Available *2007-04-28*

This release contains improvements to non-cumulative mapping, enhancements to the logic that auto discovers default field mappings, bug fixes, and feature requests.

Please check here for more release notes

1,000,000 Project Web Hits *2007-04-08*

The Dozer project recently eclipsed the 1 million hit mark. Thanks everyone! It's been fun.

Release 3.2.1 Now Available *2007-04-08*

This small release contains feature requests for Boolean to Number auto conversion, custom converter statistics, and a small performance improvement for jdk1.5 users.

Please check here for more release notes

Release 3.2 Now Available *2007-04-03*

This minor release contains bug fixes and feature requests.

Please check here for more release notes

Thanks to everyone that has submitted new bug and feature requests over the last month.

Release 3.1 Now Available *2007-03-25*

This release contains support for Java 1.5 generics (you no longer need to give hints for typed collections!), enums, performance improvements, bug fixes, feature requests, and a move to subversion. This release will also be hosted in the Maven 2 Repository.

Thanks again to everyone submitted bug and feature requests, along with everyone that contributed to the release.

Please check here for more release notes

Release 3.0 Now Available *2007-02-09*

This release contains infrastructure upgrades, bug fixes, feature requests, and some performance improvements.

One thing to note is Custom Converters are now invoked if the src value is null, so just make sure any Custom Converters you have created explicitly handle a null source value.

Thanks again to everyone submitted bug and feature requests, along with everyone that contributed to the release.

Please check here for more release notes

Release 2.4 Now Available *2006-10-16*

This release contains bug fixes and feature requests. One thing to note is that with this release RuntimeExceptions will no longer be wrapped in a MappingException. Thanks to everyone submitted bugs/feature requests and contributed to the release.

Please check here for more release notes

Release 2.3 Now Available *2006-09-01*

We are back after taking some time to enjoy the unlimited summer activities in the Rockies. This release focuses on bug fixes, feature requests, and general cleanup/refactoring of the code base. Thanks to everyone that contributed to the release!

Please check here for more release notes

Release 2.2 Now Available *2006-04-29*

This release includes runtime statistics support, JMX integration, event listening model, dozer configuration through a properties file, XMLBeans mapping, and a few other things. There is also a few bug fixes.

Please check here for more release notes

Release 2.1 Now Available *2006-04-18*

This release includes indexed mapping support, bi-directional self mapping and much more. There is also a few bug fixes.

Thank you Kiersztyn Wojtek and Peciuch Dominic for your index based mapping code contribution!

Please check here for more release notes

Release 2.0.1 Now Available *2006-02-02*

This release had a few new features and a few bug fixes.

Please check here for more release notes

Release 2.0 Now Available *2006-01-16*

This release was mainly performance based. We have seen a 400% - 800% increase in mapping performance. Check here for our profiling notes .

Please check here for more release notes

Release 1.5.8 Now Available *2005-11-30*

Removed dependency on Castor. We also added a few more features.

Please check here for release notes

Release 1.5.7 Now Available *2005-11-15*

Dozer now supports map-backed properties. This is extremely useful for mapping objects to/from many UI frameworks. The next release of Dozer will focus purely on performance enhancements.

Please check here for release notes

Release 1.5.6 Now Available *2005-10-31*

Please check here for release notes

Dozer in the Java Developer's Journal! *2005-10-15*

Dozer is mentioned as a great way to map objects between application layers in the October issue of the JDJ

About

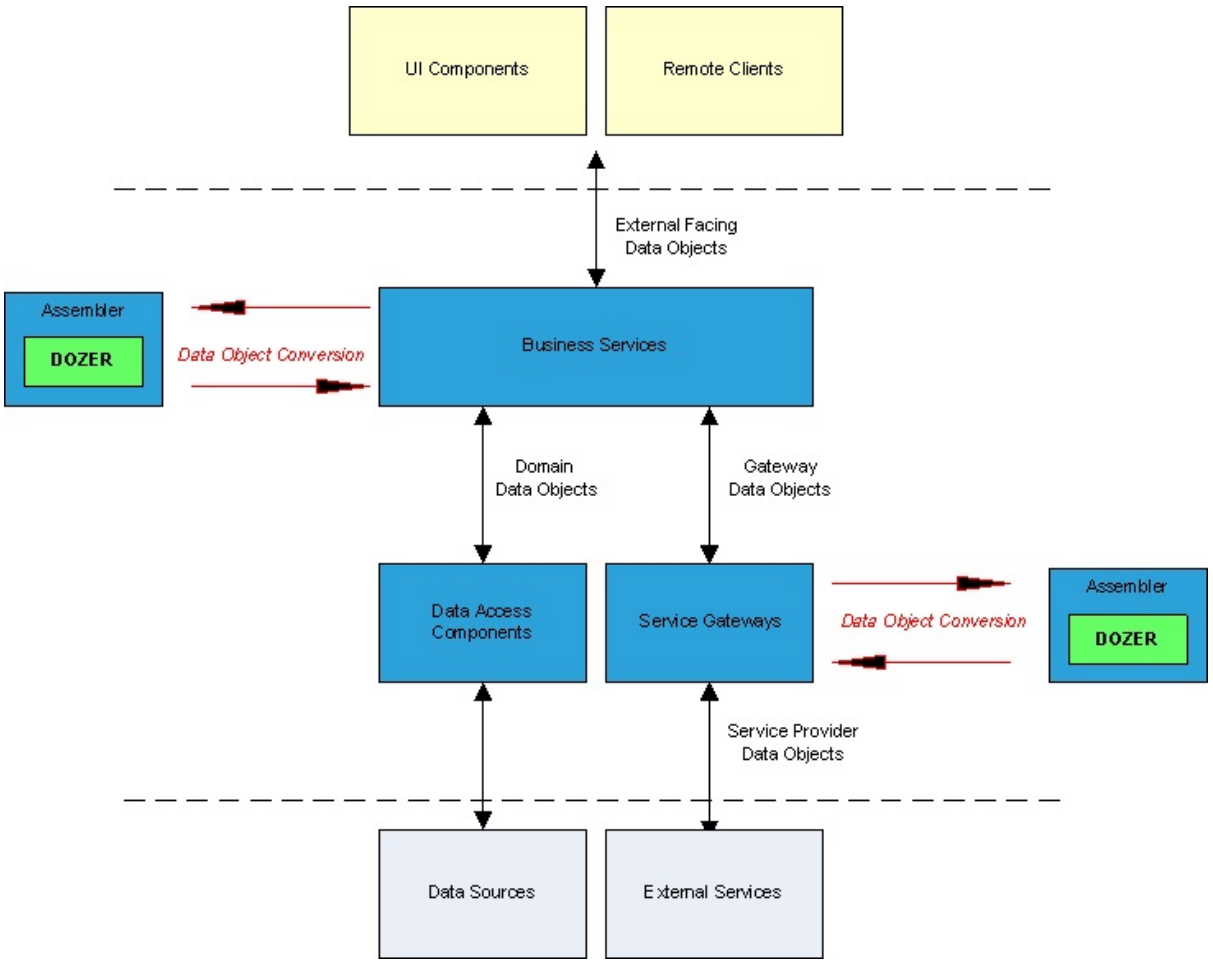
Dozer is a Java Bean to Java Bean mapper that recursively copies data from one object to another. Typically, these Java Beans will be of different complex types.

Dozer supports simple property mapping, complex type mapping, bi-directional mapping, implicit-explicit mapping, as well as recursive mapping. This includes mapping collection attributes that also need mapping at the element level.

Dozer not only supports mapping between attribute names, but also automatically converting between types. Most conversion scenarios are supported out of the box, but Dozer also allows you to specify custom conversions via XML.

The mapper is used any time you need to take one type of Java Bean and map it to another type of Java Bean. Most field mapping can be done automatically by Dozer using reflection, but any custom mapping can be predescribed in XML format. Mapping is bi-directional so only one relationship between classes needs defining. If any property names on both objects are the same you do not even need to do any explicit property mapping for these fields.

The picture below depicts some of the common areas Dozer could be inserted into an architecture. Notice that it typically is utilized at the boundaries (entry/exit). Dozer will make sure that your internal domain objects from the database do not bleed into external presentation layers or to external consumers. It can also help map your domain objects to external APIs calls and vice-versa.



The bean mapper is written in Java and relies heavily on the Jakarta Commons Bean Utils package for Java Bean utility methods.

Why Map?

Please read the Why Map? page.

Frequently Asked Questions

Please read the FAQ page.

Downloading Dozer

- Latest release of Dozer Mapper.
- Older Releases hosted on SourceForge.

License

- Apache 2.0 License

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.

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.

Getting Started

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>${dozer.version}</version>
</dependency>
```

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

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

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.

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.

```
<?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/sche
```

```
ma/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>
</mappings>
```

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.

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

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.

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 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.

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>
```

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.

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);
```

Annotation Mappings

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.

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.

```
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.

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.

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.

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.

```
import org.dozer.classmap.RelationshipType;
import org.dozer.loader.api.BeanMappingBuilder;
import org.dozer.loader.api.FieldsMappingOptions;
import org.dozer.loader.api.TypeMappingOptions;

import static org.dozer.loader.api.FieldsMappingOptions.collectionStrategy;
import static org.dozer.loader.api.FieldsMappingOptions.copyByReference;
import static org.dozer.loader.api.FieldsMappingOptions.customConverter;
import static org.dozer.loader.api.FieldsMappingOptions.customConverterId;
import static org.dozer.loader.api.FieldsMappingOptions.hintA;
import static org.dozer.loader.api.FieldsMappingOptions.hintB;
import static org.dozer.loader.api.FieldsMappingOptions.useMapId;
import static org.dozer.loader.api.TypeMappingOptions.mapId;
import static org.dozer.loader.api.TypeMappingOptions.mapNull;

public class MyClass {

    public void create() {
        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.

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/sche
ma/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.

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"

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.

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

Table 1. Dozer Configuration Properties

Property Name	Description	Vali
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
dozer.autoregister.jmx.beans	Specifies whether Dozer will auto register it’s JMX beans with the PlatformMDBServer on startup.	true
dozer.el.enabled	Specifies whether during Xml mappings parsing Dozer will recognize EL expressions.	true

dozer.cache.converter.by.dest.type.maxsize	Specifies the max size for one of Dozers internal caches.	0 - Long.M
dozer.cache.super.type.maxsize	Specifies the max size for one of Dozers internal caches	0 - Long.M
org.dozer.util.DozerProxyResolver	Specifies implementation of DozerProxyResolver to be used	Valid cl
org.dozer.util.DozerClassLoader	Specifies implementation of DozerClassLoader to be used	Valid cl

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.

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%.

Table 1. 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 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

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

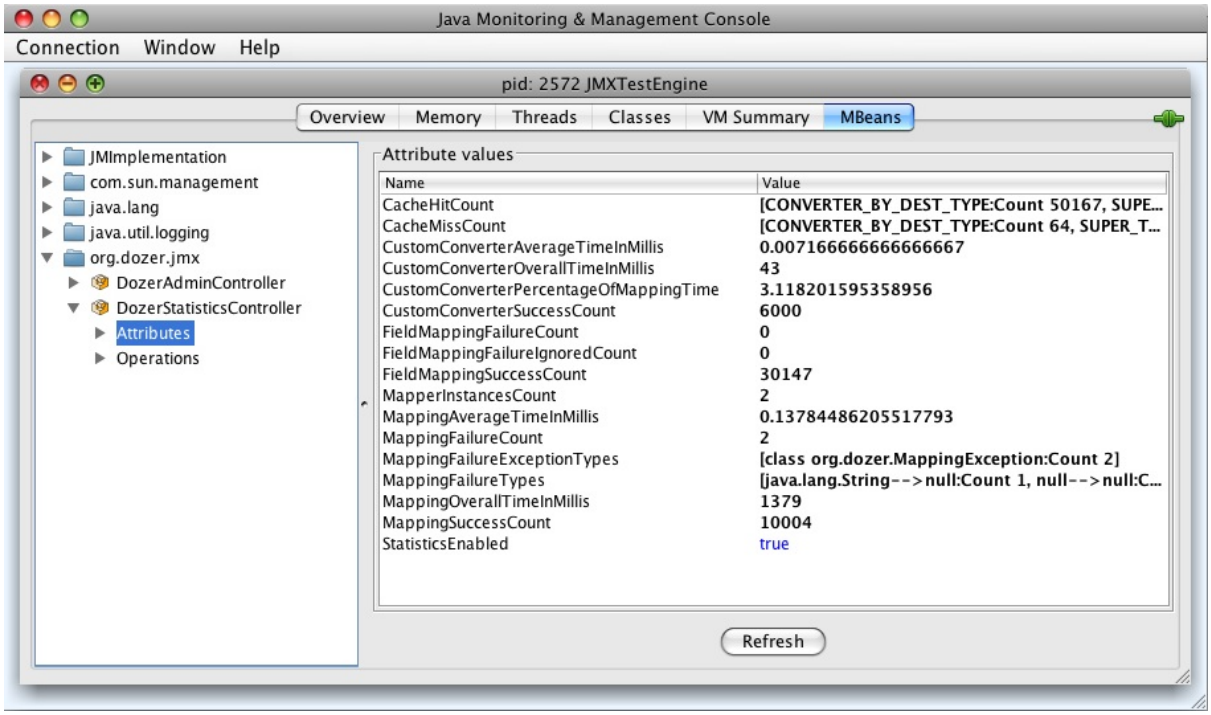
Table 1. Dozer JMX MBeans

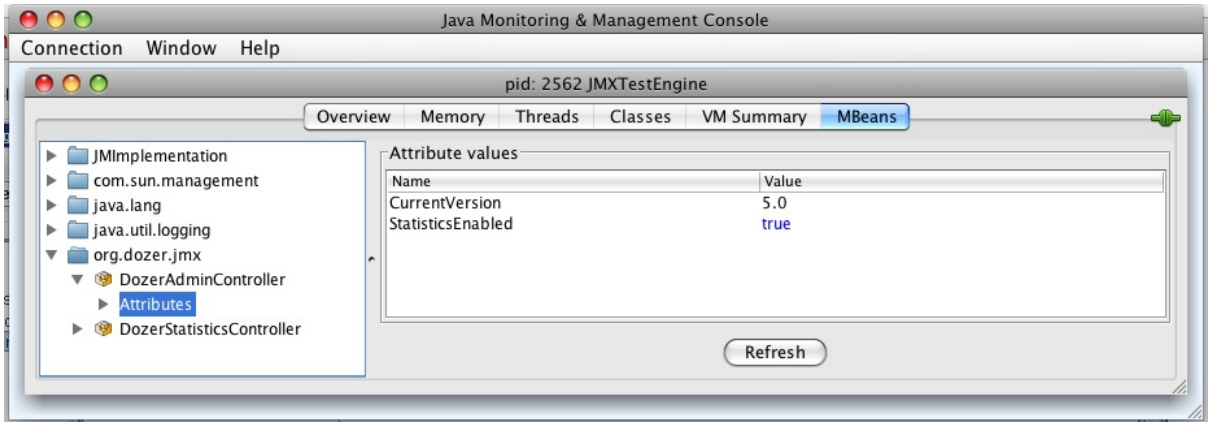
MBean	Description
DozerStatisticsControllerMBean	Runtime mapping statistics. The Statistics section contains more information on the types of statistics that are available.
DozerAdminControllerMBean	Admin functions such as enabling/disabling statistics gathering at runtime.

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.

Sample JMX Screen Shots

Dozer JMX MBeans via JConsole…..


Java Monitoring & Management Console
Connection Window Help
pid: 2572 JMXTestEngine
Overview Memory Threads Classes VM Summary MBeans
JMImplementation
com.sun.management
java.lang
java.util.logging
org.dozer.jmx
DozerAdminController
DozerStatisticsController
Attributes
Operations
Attribute values
Name Value
CacheHitCount [CONVERTER_BY_DEST_TYPE:Count 50167, SUPE...
CacheMissCount [CONVERTER_BY_DEST_TYPE:Count 64, SUPER_T...
CustomConverterAverageTimeInMillis 0.007166666666666667
CustomConverterOverallTimeInMillis 43
CustomConverterPercentageOfMappingTime 3.118201595358956
CustomConverterSuccessCount 6000
FieldMappingFailureCount 0
FieldMappingFailureIgnoredCount 0
FieldMappingSuccessCount 30147
MapperInstancesCount 2
MappingAverageTimeInMillis 0.13784486205517793
MappingFailureCount 2
MappingFailureExceptionTypes [class org.dozer.MappingException:Count 2]
MappingFailureTypes [java.lang.String-->null:Count 1, null-->null:C...
MappingOverallTimeInMillis 1379
MappingSuccessCount 10004
StatisticsEnabled true
Refresh


Java Monitoring & Management Console
Connection Window Help
pid: 2562 JMXTestEngine
Overview Memory Threads Classes VM Summary MBeans
JMImplementation
com.sun.management
java.lang
java.util.logging
org.dozer.jmx
DozerAdminController
Attributes
DozerStatisticsController
Attribute values
Name Value
CurrentVersion 5.0
StatisticsEnabled true
Refresh

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.

```
<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);
```

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.

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>
```

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.

```
<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());
```

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/sche
ma/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:

```
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.

Frequently Asked Questions

Common

Advanced

Tips, Tricks, and Suggestions

Answers

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.

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

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".

Is Dozer recursive?

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

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.

Are Collections and Arrays supported?

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

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

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

Are abstract classes, inheritance, and interface mapping supported?

Yes.

Can Dozer be configured via Spring?

Yes. Refer to Spring Integration section of the documentation.

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.

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.

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.

How are the custom xml mapping files loaded?

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

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"

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

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%.

Which JDK versions are supported?

JDK 1.8 and above.

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.

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

Is Dozer good for the environment?

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

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>
```

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>
```

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.

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

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>
```

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>
</mapping>
```

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>
```

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.

Does Dozer support JDK 1.5 enums?

Yes. Enum to Enum mapping is automatically handled.

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.

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!

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 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>
```

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

How do I enable Dozer to collect runtime mapping statistics?

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

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.

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);
  }

}
```

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.

```
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);
}
```

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.

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.

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.

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/target/test-classes/doze
r.properties] for Dozerglobal property configuration
dozer: Reading Dozer properties from URL[file:/local/subversion_projects/dozer/trunk/t
arget/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=DozerStatisticsController] auto registered
with the Platform MBeanServer
dozer: Dozer JMX MBean [org.dozer.jmx:type=DozerAdminController] auto registered with
the Platform MBeanServer
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 insta
nce:[fieldAttributeMapping.xml]
dozer: Trying to find xml mapping file: fieldAttributeMapping.xml
dozer: Using URL [file:/local/subversion_projects/dozer/trunk/target/test-classes/fiel
dAttributeMapping.xml]to load custom xml mappings
dozer: Successfully loaded custom xml mappings from URL:[file:/local/subversion_projec
ts/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….

```
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
```

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/sche
ma/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>
```

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 GitHub project page

Global and Map Level Configuration

The configuration block is used to set the global defalut 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. Each of these mapping files can have their own configuration block. A mapping will inherit its configuration from the mapping file that it is stored in. **Implicit mappings will inherit the default values for configuration.**

The following is the sample configuration block from the example mappings file:

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

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>
```

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>
```

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" 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>
```

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:

```
<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>
```

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…

```
<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>
```

One-Way and Bi-Directional 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.

```
<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>
<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>
</mapping>
```

Mapping Concepts

Assembler Pattern

Dozer can be used as an Assembler . Martin Fowler has a great explanation of why and when you would use an Assembler . Basically, it is a way to take multiple fine grain objects and create one coarse grain object used for data transfer. As long as you have mappings defined for each of your fine grained objects to your coarse grain object you can call the mapper multiple times to achieve the desired assembler pattern.

```
mapper.map(sourceA, ClassB.class);
```

Let’s say that ClassA, ClassB, and ClassC all map to ClassD. First create these individual mappings in the mapping file. Next, call the mapper once for each mapping. Note that this would also work in a bi-directinonal manner.

```
ClassD d = new ClassD();
mapper.map(sourceA, d);
mapper.map(sourceB, d);
mapper.map(sourceC, d);
```

Inheritance Mapping

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:

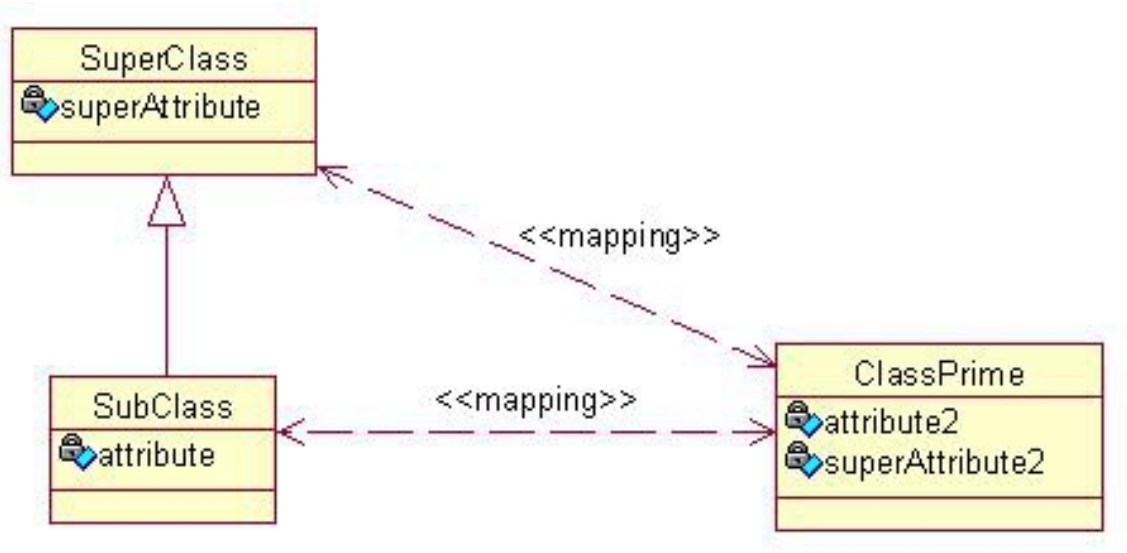
```
<mapping>
    <class-a>org.dozer.vo.SubClass</class-a>
    <class-b>org.dozer.vo.SubClassPrime</class-b>
    <field>
        <!-- this is the same for all sub classes -->
        <a>superAttribute</a>
        <b>superAttr</b>
    </field>
    <field>
        <a>attribute2</a>
        <b>attributePrime2</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.

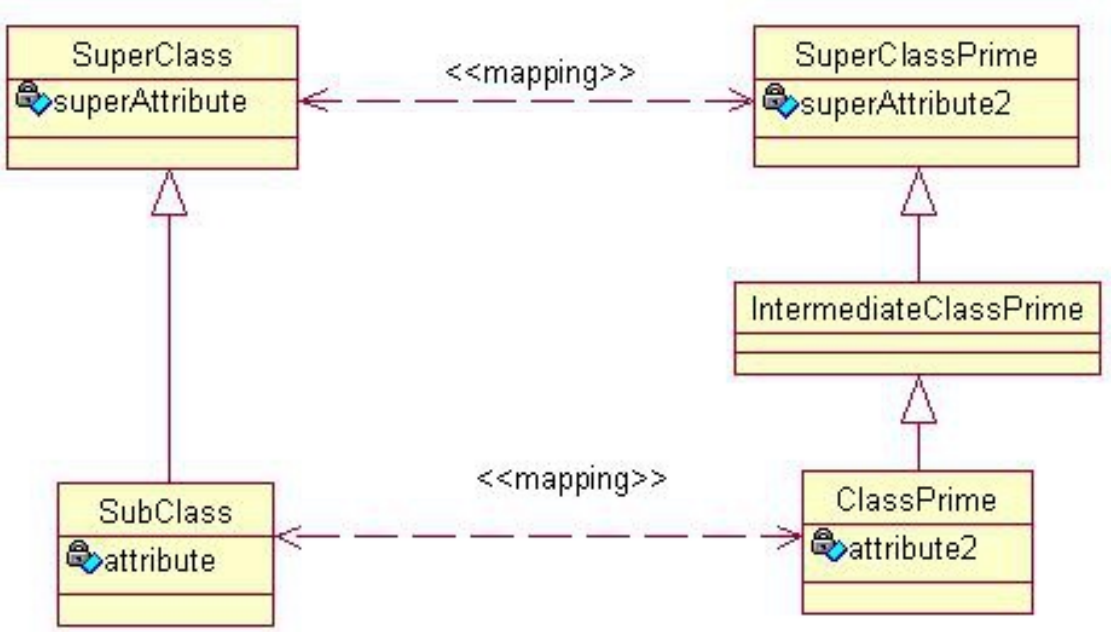
However, a better way to do it would be to map the base class individually. This can be done for each base class (in the case of a larger heirarchy). Assuming the base class name, below is the refactored mapping xml:

```
<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>
```

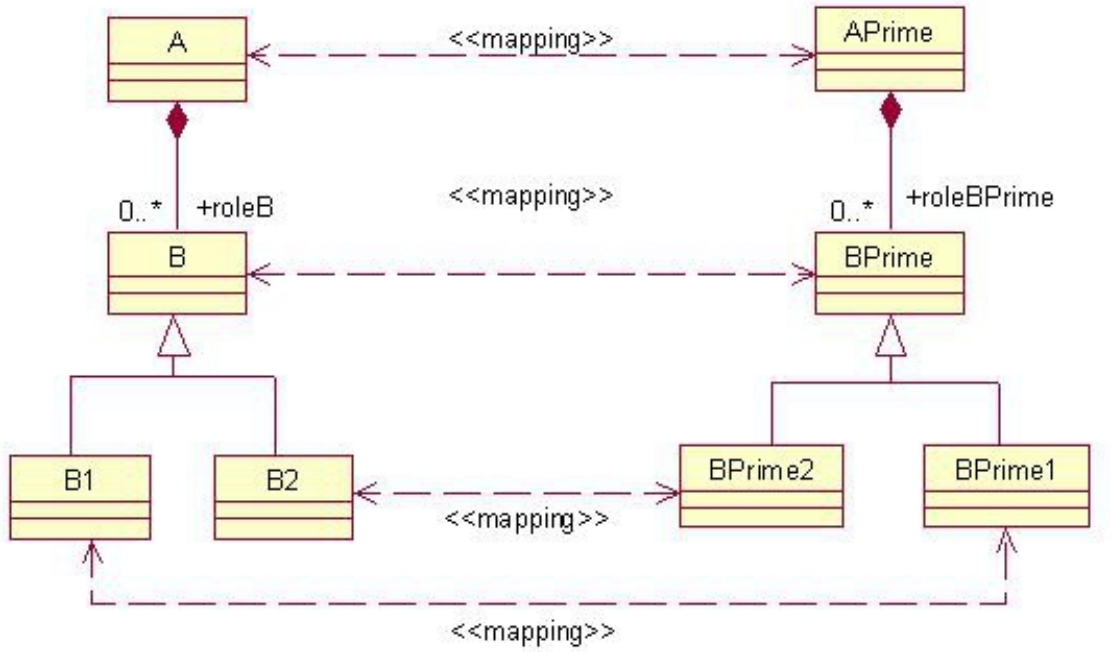
The following images explain some of the different scenarios dozer handles. Each diagram shows two mapped class hierarchies and existing relations, which Dozer recognizes and maps.



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.



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>
```

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

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
- 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

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.

```
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;
  }

}
```

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.

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

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>
```

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….

```
<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.

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.

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

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>
```

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

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

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.

```
<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>
```

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>
```

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.

```
<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>
```

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>
```

Spring bean factory injection

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

```
<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.SampleCustomBeanFact
ory" />
</beans>
```

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

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/sche
ma/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.

```
<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.

```
<?xml version="1.0" encoding="UTF-8"?>
<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="configurableConverterBeanInsta
```

```
nce1" />
                <entry key="CustomConverterWithId2" ref="configurableConverterBeanInst
ance2" />
            </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.

```
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.

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

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>
```

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>
```

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-conve
rter-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.

```
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();
    }
  }

}
```

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. 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.

```
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.

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.

```
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;
  }

}
```

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>
```

Custom get() set() Methods

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>
```

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>
```

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.

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

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>
```

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.

```
<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>
```

Deep Indexed Mapping

Indexed mapping within deep mapping is supported.

```
<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>
```

JDK 1.5 Enum Mapping

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

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

```
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;
  }

}
```

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:

```
<?xml version="1.0" encoding="UTF-8"?>
<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>
```

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>
```

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`.

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:

```
<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>
```

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…

```
<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>
```

Expression Language

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>
    </mapping>
</configuration>
```

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.

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>
```

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>
```

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>
```

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>
```

Overriding Trim Strings Policy

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

```
<!-- 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>
```

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>
```

Map Backed Property Mapping

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>
```

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>
```

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="myCustomNullProper
ty">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>
```

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.

```
<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.

```
// 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());
```

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.

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>
</mapping>
```

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.

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

Proxy Objects

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.

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.

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.

Basic Property Mapping

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.

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>
```

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.

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.

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

```
<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>
```

Examples

There are some sample mapping files under \{dozer.home}/src/test/resources. These mapping files are used by the Dozer unit tests.

Prerequisites

The Plugin currently depends on

- WTP 3.0.2 or latest

Compatibility

Plugin is supported and been tested on the following IDEs

- Eclipse 3.6
- RAD RSA 8.0.1

Installation

The Dozer Plugin-feature can be installed and updated using the Eclipse Update Manager.

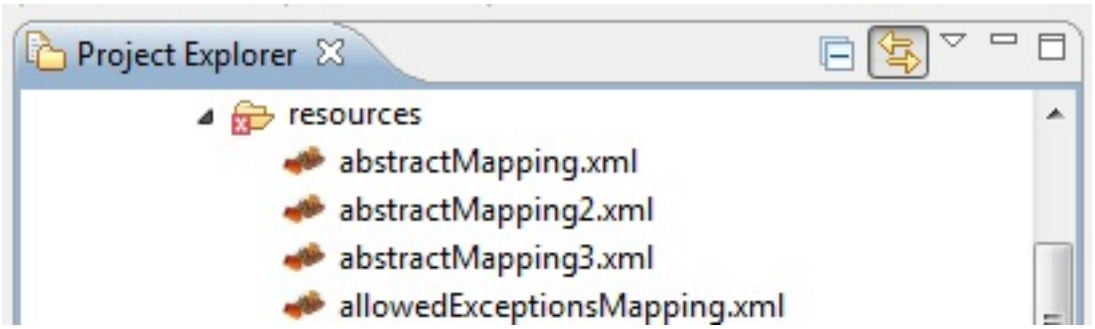
- Add a new Update-URL:

```
http://dozer.sourceforge.net/eclipse-plugin
```

Select all required dependencies and install the plugin.

Alternatively you can download the package at sourceforge and unzip it in your eclipse installation folder. You might have to enable the Plugin after starting Eclipse. This can be done at Help > Software Updates > Manage Configuration.

After installation is done you should see a little red dozer-icon on all your mapping-xml files.



Create new Mapping files

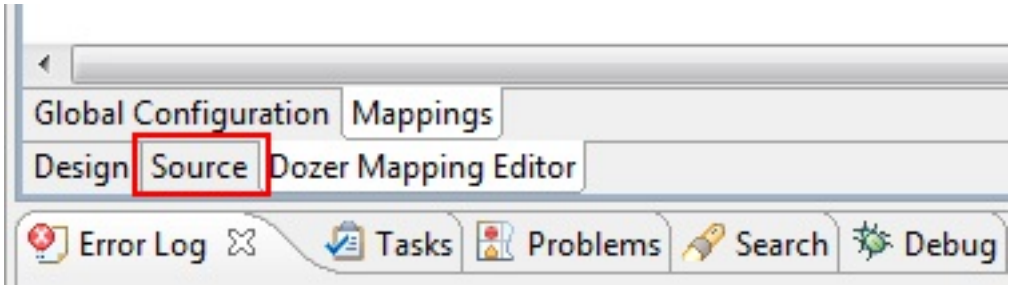
The plugin provides a simple wizard for creating new mapping-xml files. A new mapping-xml file can be created using File > New > Other > Dozer Mapping Framework > Dozer Mapping File. Select the correct dozer version (4, DTD or 5, XSD) in the wizard and create the file.

Configure the Mapping

- via XML
- via the Editor

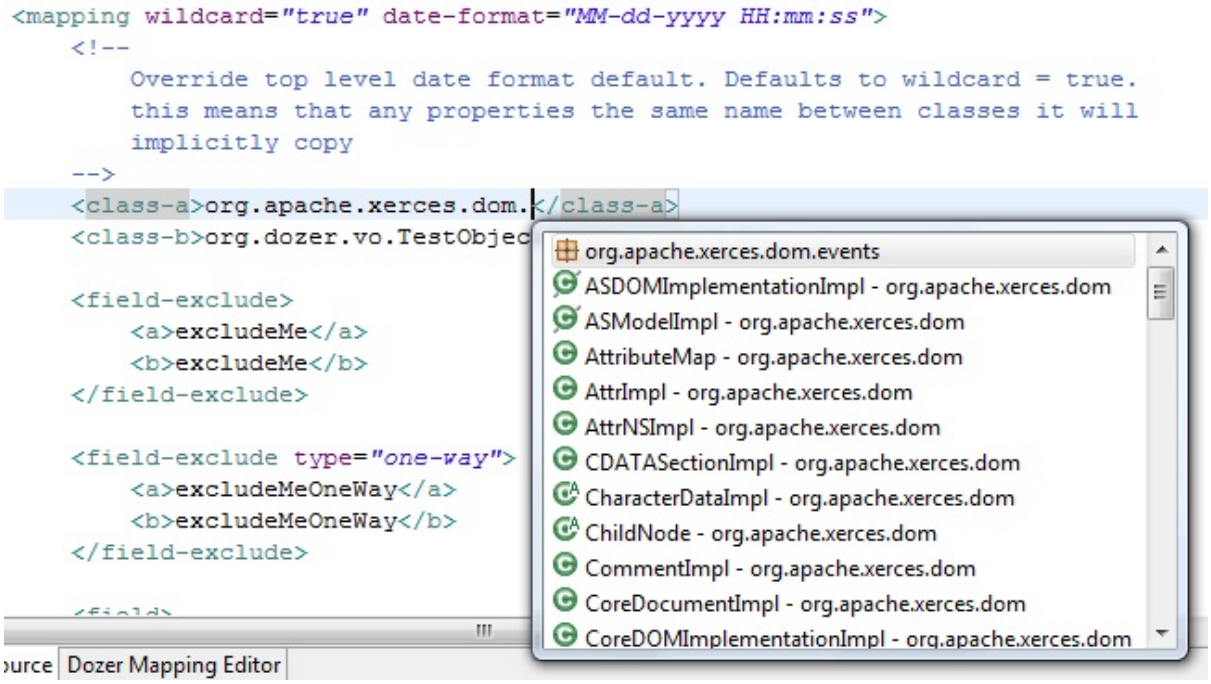
Open mapping file

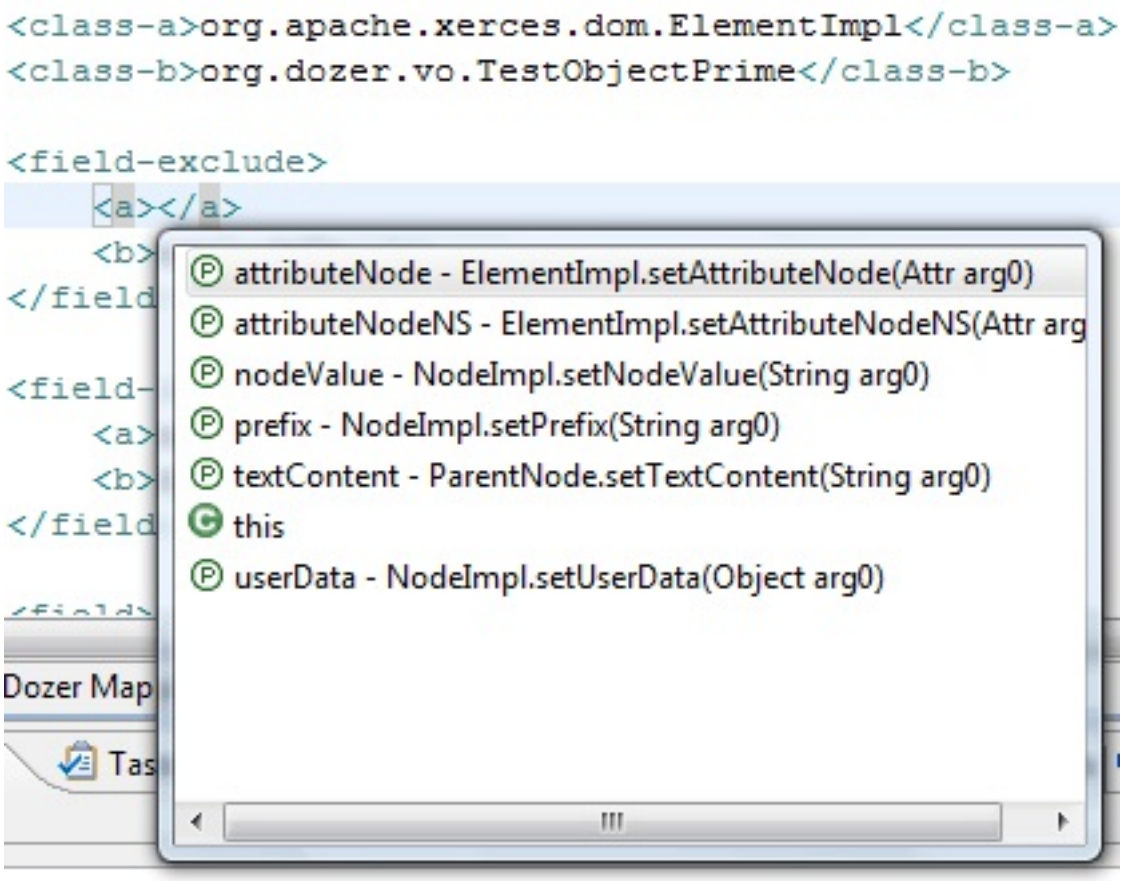
When opening a Dozer-mapping-file the Dozer-editor appears. If you want to edit the raw XML you can switch to the "source" view.



Content Assist

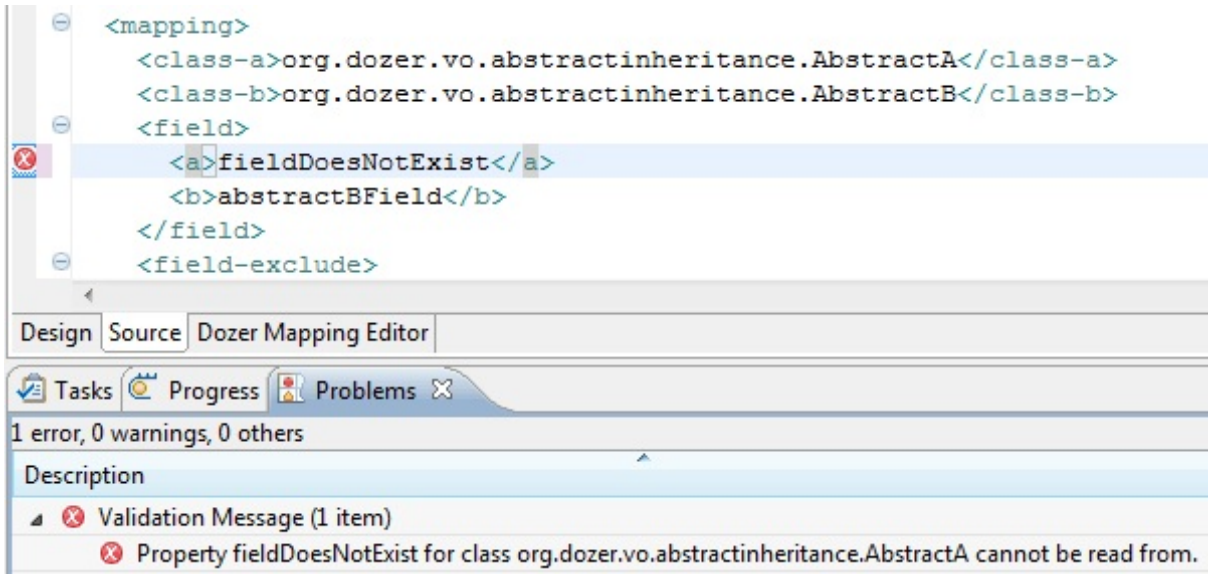
All the class-a/class-b, field, get-method, set-method, etc -nodes or -attributes automaticly show content-assist popups when pressing ctrl+space or the configured WTP-XML content assist characters. These can be changed in the preferences.





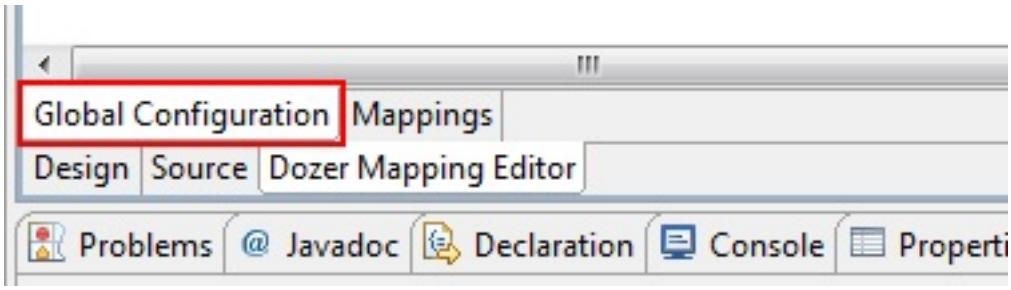
Validation

The Dozer Plugin validates the Mapping to find out if the mapped class do exist and if the mapped fields are accessible.



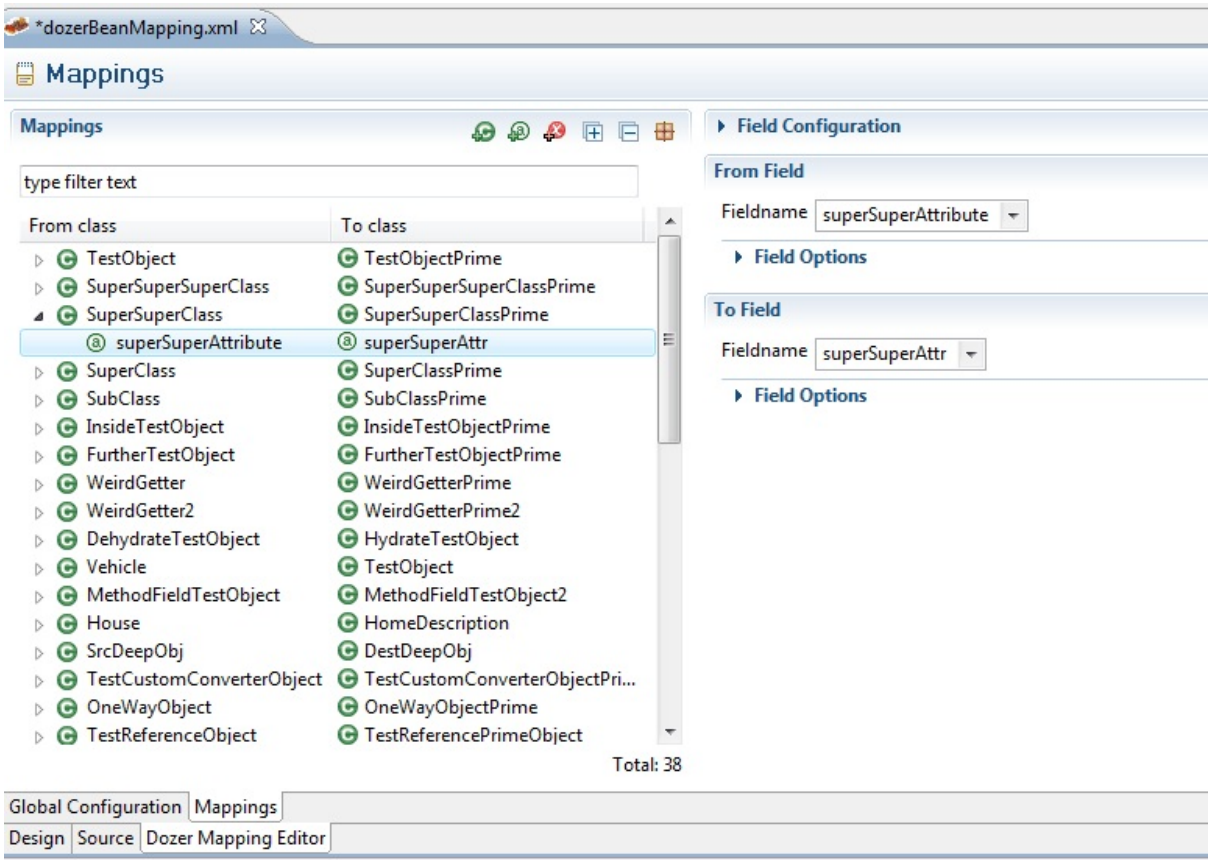
Setting global dozer configuration

All global configuration values can be set in the "Global Configuration" Tab.



Configuring mappings

Every classmapping is listed in the "Mapping" Tab. Mappings of classes and fields can be added by Popup menu or the action-buttons at the top. On the right side the mappings can be configured. Every Attribute or Element in the XML file is shown for editing.



Articles and Tutorials

This page aggregates links to external articles, blog posts and tutorials related to Dozer.

- Automated Bean Conversion
- Dozer Mapping
- Dozer Type Conversion in Apache Camel
- Using Dozer with GXT
- Simple Dozer Example
- Using Dozer with Hibernate and GWT
- Java Bean Mapping with Dozer

Support

Looking for support or want to propose an enhancement for the dozer Framework?

For community support or dozer enhancement requests check out the Google Group . We value your enhancement suggestions and want to build something of value to the Java community. When submitting bug and feature requests, it is really helpful if you are able to submit a simplified unit test case(s) that reproduce the issue. We understand this is not possible for every request, but if you are able to submit a unit test it usually results in the request being implemented sooner.

Think you've found a bug?

Please enter a bug in the GitHub Issue Tracker. We value your support there. We can't fix bugs we don't know about!

Have a feature request?

Please enter a feature request in the GitHub Issue Tracker. It helps if you attach a pull request :)

Got a patch?

Please submit a pull request for Dozer GitHub project. Along with your pull request we need a unit test and any supporting objects and mapping file.

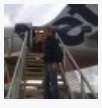
Project Team

A successful project requires many people to play many roles. Some members write code or documentation, while others are valuable as testers, submitting patches and suggestions.

The project team is comprised of Members and Contributors. Members have direct access to the source of a project and actively evolve the code-base. Contributors improve the project through submission of patches and suggestions to the Members. The number of Contributors to the project is unbounded. Get involved today. All contributions to the project are greatly appreciated.

Members

The following is a list of developers with commit privileges that have directly contributed to the project in one way or another.

Image	Id	Name	Email	Rol
	fgarsombke	Franz Garsombke	fgarsombke@yahoo.com	Proje Lead/ Foun
	mhtierney	Matt Tierney	mhtierney@comcast.net	Proje Lead/ Foun
	buzdin	Dmitry Buzdin	buzdin@gmail.com	Proje Lead
	alankstewart	Alan Stewart	alankstewart@gmail.com	Devel

Contributors

The following additional people have contributed to this project through the way of suggestions, patches or documentation.

Name
Alan Stewart
Ben Sullins
Bruno Guedes
Chia-Chou Hung
Christoph Goldner
David Baker
Dmitry Spikhalskiy
Ed Bras
Florian Kammermann
Gerard Toonstra
Hee Tatt Ooi
James Bassett
Joachim Korittky
Kiersztyn Wojtek
Knut-Erik Johnsen
Lenkwe Makhubela
Luke Robinson

Mariusz Pala
Markus Thielen
Matt Benson
Mickael Morier
Nhat Vo
Ozzie Gurkan
Peciuch Dominic
Peter Monks
Rob Volden
Rohan Hart
Sachin Katakdound
Soren Chittka
Tim Nee
Tom Rigole
Vadim Shaigorodskiy
Vincent Jassogne
Wing Tung Leung

Issue Management

Issues, bugs, and feature requests should be submitted to the following issue management system for this project.

```
https://github.com/DozerMapper/dozer/issues
```

Source Repository

This project uses Git to manage its source code. Instructions on Git use can be found at http://git-scm.com/documentation.

Web Browser Access

The following is a link to a browsable version of the source repository:

```
https://github.com/DozerMapper/Dozer/dozer-plugins-parent/dozer-site
```

Anonymous Access

The source can be checked out anonymously from Git with this command (See http://git-scm.com/docs/git-clone):

```
$ git clone https//github.com/DozerMapper/dozer/dozer-plugins-parent/dozer-site
```

Developer Access

Only project developers can access the Git tree via this method (See http://git-scm.com/docs/git-clone).

```
$ git clone git@github.com:DozerMapper/dozer.git
```

Access from Behind a Firewall

Refer to the documentation of the SCM used for more information about access behind a firewall.

Release Notes

2017-04-19 Moving to github releases

Its all on github!

Archived Release Notes

See for pre 2017 release notes.

Release Notes

5.5.1 *2014-04-22*

Release Notes on GitHub

5.4.0 *2012-12-01*

Bug Fixes

- Mapping null values within List to List mapping causes NPE link
- Multiple superclass mappings, fields mapped over and over link
- Deep index mapping for untyped Collection link
- Iterate mapping with more than one hint fails link
- Inheritance problems with custom-converters on fields link
- Fix for generic introspection issues link
- Does not swallow InterruptedException link

Feature Requests

- Migration to GitHub
- Web Site Update
- Mapping Metadata Introspection link
- Added possibility to convert Enums from Strings link
- Load mappings via InputStream link

5.3.2 *2011-02-15*

Bug Fixes

- NullPointerException when mapping single field to List link

- StackOverFlow when using MapperAware Custom Converters link
- ClassMappings get wrong ClassMap in polymorphism case link
- Mapping of XMLGregorianCalendar not working link
- Thread hang during initialization link
- Add ability to define is-accessible at the type level link
- one-way doesn’t work at the mapping level link
- Mapping String collections with "iterate" fails link

Feature Requests

- Get a handle on mappedFields property from CustomConverter. link
- BigDecimal and Double mapping link
- Integrate BeanUtils 1.8 converters link

5.3.1 *2010-11-15*

Bug Fixes

- DozerBeanMapper initialization is not thread safe link
- Collection Generic type not resolved on field levelhttps://sourceforge.net/tracker/?func=detail&aid=3066742&group_id=133517&atid=727368[link]

Feature Requests

- Apache Camel Integration Improvement link

5.3.0 *2010-10-10*

Bug Fixes

- ReflectionUtils.findAMethod can escape by returning null link
- Unnecessary call to a method link
- XMLGregorianCalendar instantiation fails link

- DateFormat instantiation fails link
- Mixed-type values in Map are incorrectly converted link
- Fail to map an object graph correctly link
- Semi-undocumented dependency on XMLBeans link
- Bug with Javassist link
- Bug in TypeResolver with generics/inheritance link
- Remove-orphans is not handled when mapping java.util.Map link
- Hard to use XML config when mapping from/to java.util.Map link
- Deep property mapping and is-accesssible=true incompatible link

Feature Requests

- Alternative Java API for mappings link
- Commons Logging replaced with SLF4J link
- Improved Hibernate proxy handling link
- Warning on duplicate mappings link
- Improved generics handling link

5.2.2 2010-06-09

Bug Fixes

- CustomConverterContainer puts null results in cache link
- NPE in SimpleElementReader link
- DozerConverter fails on primitive to primitive map link
- T extends Enum<StatisticType> link
- Statistics are not thread safe link
- Copy-by-reference ignored when object is in a Set link
- ConcurrentModificationException on the first map link

- Mapping primitive List → primitive array does not work link

Feature Requests

- Allow static createMethod on any class link
- Improve debug logging link

5.2.1 *2010-04-28*

Bug Fixes

- Forgotten Custom Converter Parameter in Map-backed Mappings link
- DozerConverter does not Check for Runtime Types Properly link
- Support for XMLGregorianCalendar link
- Support of Map implementations other than HashMap link

Feature Requests

- Dozer OSGi Bundle link
- Expression Language Support (javax.el) link
- Drop Dependency on Apache Commons Collections link

5.2.0 *2010-01-24*

Bug Fixes

- Failing Top Level Mapping (this → this) link
- NPE on Missing Getter link
- Exception Message Imrovement link
- Iterate Mapping does not Check Previously Mapped Objects link
- Stack Overflow on Recursive Object Graphs link
- Race Condition in Statistics Area link
- Date to Calendar Convertation link

- Stored References to Generated Classes link
- Fail on null Value in Set link
- Exception on Mapping of XMLGregorianCalendar link
- Non-Standard getter/setter names and deep-mapping issue link
- Fix for null Element in Array link

Feature Requests

- Support for Spring Resources link
- Support for Collections without Setters link
- Custom Converter with Injected Mapper link

5.1 *2009-08-25*

Bug Fixes

- JAXBBeanFactory can not create bean for nested type link
- Mapping definition of super interface is ignored link
- Inheritance mapping with proxy object link
- Processing Map with Null entry fails link
- Date Format is not used for collection elements link
- Memory Leak on Web Container reload when using JMX Beans link
- Hints does not work for inherited classes link
- Null values are not copied to destination Map link
- Map with List entries is mapped incorrectly link
- NPE when missing one of the class names in mapping definition link

Feature Requests

- Support for Generic return types link
- Generics in Custom Converter API link

- Support for custom proxy resolver link
- Support for custom class loaders link

5.0 *2009-03-03*

Migration Guide

Change Description	New - Dozer 5.0	
Increased minimum JDK requirements	Requires JDK 1.5+	Required JDK 1.4
Repackaged Dozer classes	org.dozer	net.sf.dozer.util.m
Maven group id	net.sf.dozer	net.sf.dozer
Added Generics to public api	A dest = mapper.map(b, A.class)	A dest = (A) mapp
Switched to XSD instead of DTD	beanmapping.xsd	dozerbeanmappi
Renamed some public Interfaces	BeanFactory	BeanFactoryIF
	CustomFieldMapper	CustomFieldMap
	Mapper	MapperIF
Repackaged some public Interfaces	org.dozer.CustomConverter	net.sf.dozer.util.m

	org.dozer.ConfigurableCustomConverter	net.sf.dozer.util.m
	org.dozer.DozerEventListener	net.sf.dozer.util.m
Upgraded 3rd party dependencies	commons-collections 3.2.1	commons-collecti
	commons-beanutils 1.8.0	commons-beanut
	commons-lang 2.4	commons-lang 2.
	commons-logging 1.1.1	commons-logging
	junit 4.5	junit 3.8

+

Bug Fixes and Patches

- DozerResolver should resolve the DTD using class classLoader link
- Inheritance + Map-Id link
- Exclude test classes from the dist jar link
- Timestamp mapping broken link
- Map method lookup link
- DozerBeanMapper doesn't resolve graphs properly link
- Bug using MapIds link
- Bug with MapIds using Hints link

Feature Requests

- Change maven group id link
- Repackage dozer classes link

- Add Generics to public Mapper interface link
- Upgrade to JUnit 4 link
- Upgrade 3rd party dependencies link
- Switch to XSD instead of DTD link
- Migrate to JDK 5 link
- Rename some public interfaces link

4.4.1 *2009-01-31*

Bug Fixes

- Using ContextClassLoader to load classes link

4.4 *2008-12-27*

Bug Fixes and Patches

- Global Configuration is Ignored link
- Throw exception if more than one global configuration found link
- Problem loading Dozer bean mapper from Spring link
- Error in loading mapping files located in jar file link
- NullPointerException link
- Problem with Java 1.5 enum and inheritance link
- Deep Index Custom Converter Problem link
- Bug fix not indexed properties (List - Vector) link
- Deep index patch link
- Collection deep index issue link
- Deep Mapping with custom setter method does not work when default setter exists link

- Improve an exception message link

Feature Requests

- Reorganize test mapping and config files in src tree link

4.3 *2008-12-03*

Bug Fixes

- Class hierarchies are not mapped in descending order link
- Map-backed properties are not copied using deep mapping link
- Mapping with remove-orphans changes the mapped list order link
- Mapping with remove-orphans should call remove() on the list link
- Bug in ReflectionUtils.getInterfacePropertyDescriptors() link
- Classloading approach is not recommended link
- NullPointerException when mapping a map link
- Subclasses are not recognized link
- Mapping with parent-child relation does not work link
- Missing destination read method throws NPE link
- MappingProcessor does not use hints from DestHintContainer link

Feature Requests

- Backport to commons-collections 3.0 link
- Mask in copy-by-reference link
- Support for Javassist proxy objects link
- Config parameter for Custom Converter link
- Fix FindBugs issues link
- Documentation update link

4.2.1 *2008-06-22*

Bug Fixes

- stop-on-errors bug 1953945

4.2 *2007-12-16*

Bug Fixes

- Update spring custom converter documentation 1841449
- Trim Strings issue with data type conversion 1841448
- Orphans not removed from Sets 1822421
- TimeZone not copied when mapping Calendar 1815199
- Problem when getters and setters are on different interfaces 1814758
- Unexpected exception in LogMsgFactory 1797808
- Map-backed mapping with map-id 1796344
- Custom converter called on null source field 1792048
- Patch for determining source field type 1823435

4.1 *2007-09-22*

Bug Fixes

- Inheritance issue(s) with proxied data objects 1777357
- Array mapping with XMLBeans throws NoSuchMehtod exception 1773425
- Mapping Started Event Not firing on mapping 1768660
- Inheritance mapping not working correctly - Part Deux 1757573
- Test Cases section not diplaying correctly in web site docs 1657562

Feature Requests

- Update inheritance mapping doc's 1778316

- Repackage functional/e2e tests 1777076
- Using multiple instances of CustomConverter 1770440
- Dozer vs. XmlBeans generated primitive types 1764916
- DozerEvent - sourceObject always null. 1762642
- Path to ResourceLoader to accept any URL 1757321
- Run existing tests in both unproxied and cglib proxied mode 1756584
- Removal of orphans in destination collection 1755838
- Allow global configuration of relationship-type 1750158

4.0 *2007-07-15*

Bug Fixes

- Field custom converter bug with Map data types 1749982
- Prob w/ map-get-method and date-format 1733793
- Global custom converters missing in default ClassMap 1728385
- Copy Reference mapping instructions ignored for subclasses 1728159
- mvn eclipse:eclipse does not get all dependencies 1727717
- Prob w/ map-get-method and CustomConverter 1724104

Feature Requests

- Misc code cleanup and refactoring 1754179
- Refactor/Clean Up Dest Bean Creator 1752379
- Mapping deep level field in Indexed structure 1752329
- Add map-id to field mapping debug output 1750157
- Push mapping value hierarchy down to ClassMap and FieldMap 1749805
- Add bidirectional relationship between FieldMap and ClassMap 1749804
- Don’t expose DozerField and DozerClass objects 1749802

- Remove **PRIME** feature from the docs 1736864
- Major refactor of Map backed properties 1734665
- Repackage ClassMap.java 1734228
- Load classes consistently 1734163
- Cleanup of internal exception handling 1734161
- Add config support for auto trimming of strings 1707034
- Mapping deep level field in Indexed structure 1473800

3.4 *2007-05-19*

Bug Fixes

- Incorrectly recognizing JDK 6.0 1717547
- Null pointer on MappingProcessor at Line 282 1717318
- Change the way we determine JDK Version 1715819
- Lost and Duplicated Objects 1715496
- Fix Map VO with no custom mappings 1713550
- Propagate exceptions while parsing allowed-exceptions xml 1713242
- NPE when Date String when no date format specified 1711580
- inappropriate subclass mappings applied 1674199
- Inheritance mapping not working correctly 1486105
- bidirectionnal mapping with sets subclasses 1664984

Feature Requests

- Change util classes to static 1713604
- Add PMD and Findbugs reports 1712886
- Remove NotFoundException and DozerRuntimeExceptions 1712513
- Add class level javadoc for classes missing it 1696636

3.3.1 *2007-04-28*

Bug Fixes

- Remove Spring runtime dependency. Revert back to using Apache Commons to get Property Descriptors 1709117

3.3 *2007-04-26*

Bug Fixes

- Throw exception if map-id cannot be resolved 1706291
- When adding default field mappings, skip getter's w/params 1705525
- When discovering default field mappings require corresponding get/set method 1704085
- Global Bean Factory not applied to default mappings 1700448
- Non-Cumulative mapping issues 1698069

Feature Requests

- Use Spring's BeanUtils.getPropDescriptors() instead of jakarta 1707014
- When auto registering mbeans check if mbean is already reg'd 1697294
- Recursive object mapping not working with interfaces 1658168
- Improve collection handling, esp. non-cumulative mapping 1482749

3.2.1 *2007-04-08*

Feature Requests

- Boolean to number auto conversion 1695408
- Added statistics for custom converter mappings 1695380
- Small performance improvement for jdk1.5 users 1694734

3.2 *2007-04-03*

Bug Fixes

- 3.1 Release not backwards compatible with JDK 1.3 1692620
- Destination Value always null in CustomConverter 1679996
- Indexed Mapping broke when is-accessible is true 1673152
- Allowed Exceptions not working for default mappings 1658569
- Remove is-accessible option from configuration and mapping sections in the DTD 1692603
- Set mapping problem when field starts in upper case in mapping xml file 1549738

Feature Requests

- Auto register Dozer JMX MBeans with the platform mbean server 1690327
- Private constructor support when creating new instances of data objects 1690298
- Create quick reference page for mapping xml options 1657611
- Set/get method for last field in deep-chaing 1456486
- Add documentation for existing custom converter support of Array types 1691021

3.1 *2007-03-25*

Patches

- Bi-direction is-accessible 1664865

Bug Fixes

- Dramatic Performance Degradation with Interfaces 1684237

Feature Requests

- Transparent Java 5 enums support 1685083
- Move dozer to subversion 1684934

- Maven 2 Repo 1677946
- Resolve collection hints from Java 5 generic type information 1677376

3.0 *2007-02-08*

Bug Fixes

- Fixed custom converter cache 1644966
- Fixed wildcard mapping not working with interface inheritance 1636354
- Fixed inherited setters not found with interface inheritances 1637106
- Fixed mapping from object to array 1616229
- Fixed ClassMapFinder does not find ClassMap for interfaces 1615377
- Added Spring FactoryBean 1613791
- Fixed is-accessible problem with abstract super classes 1599457
- Fixed ClassMap not found for interface mapping 1554793
- Fixed issue with custom converters not being invoked for null values 1563795

Feature Requests

- Upgrade build infrastructure to Maven2 1651498
- Removed dependency on log4j 1644537
- Misc performance improvements 1645687
- Cleaned up indexed logic and custom converter logic 1620589
- Prevent infinite loop for bi-directional data object relationships 1596766
- Added support for custom converters at field level 1476780
- Modified custom converting matching logic 1481357
- Added support for custom field mappers 1654784

2.4 *2006-10-14*

Bug Fixes

- Added ability to load custom mapping files from outside of classpath 1563130
- Runtime exceptions no longer wrapped in MappingException 1561837
- Fixed primitive array to List mapping issue 1561184
- Fixed ConcurrentModificationException 1550275
- Fixed Proxy/Hibernate Lazy Init Object Issues 1572949

Feature Requests

- Request for passing up RuntimeException when stop on error is set to false 1513128
- Request for JAXB object factory 1572996
- Request for allowing alternate declaration of mapping files 1480372
- Request for proper null conversion for "proxy" mappings 1471808
- Request for Mapping deep level field in Indexed structure 1473800

2.3 *2006-09-01*

Bug Fixes

- Fixed String to indexed Set using a destination hint 1543202
- Fixed duplicate map-id's found 1539461
- Fixed unable to map SortedSet subinterface 1538441
- Fixed source property desciptor caching 1537668
- Fixed isAccessible not being able to find private fields 1503670
- Fixed index mapping with Set 1480666
- Attempted to fix ConcurrentModificationException 1550275

Feature Requests

- Request for new logging category for startup/init information 1475235

- Request for bypass of set dest value when the dest value already equals src value 1481427
- Request to use static map of threadsafe primitive + wrapper converters 1481500
- Request to make DozerBeanMapper.getMappingProcessor() protected 1470425
- Request for improve testability/readability of the code base 1543302
- Request to remove SourceField, DestinationField classes 1480804
- Request to remove SrcClass, DestClass, SrcHint, DestHint, and Hydrate classes 1539455
- Request to move duplicate assemble key logic to common place 1481505
- Request to add more unit testing around JMX controller classes 1481538
- Request to make map-id more intuitive from a coding perspective 1484397
- Request to clean up PMD errors 1539639
- Request to add more comments to the code base 1543264

2.2 2006-04-29

- Request Implement Event Listening Model RFE 1470590
- Request Remove Hydrate and Dehydrate mapping code RFE 1474422
- Request Remove mapping with just source object RFE 1474413
- Request Mapping XmlBean to JavaBeans Objects RFE 1468926
- Request DozerBeanMapper extensibility RFE 1470425
- Request field-exclude does not support type one-way RFE 1474440
- Fixed Duplicate Class Mapping Found Issue 1477786
- Fixed custom converter(s) and primitive matching Issue 1474216
- Fixed CGLIB Source Class - object not instance of declaring class Issue 1427982
- Added JMX integration and hooks. RFE 1475229

- Added runtime Statistics support. RFE 1475228
- Added support for Dozer configuration via properties file. RFE 1475232
- Cleaned up project documentation. RFE 1470452

2.1.1 *2006-04-18*

- Fixed performance degradation.

2.1 *2006-03-15*

- Refactored the code extensively to support a more flexible property descriptor model
- Request Index based mappings http://sourceforge.net/tracker/index.php?func=detail&aid=1468873&group_id=133517&atid=727371[RFE 1468873. Thank you Kiersztyn Wojtek and Peciuch Dominic for your code contribution!
- Request Make 'this' keyword functionality bi-directional RFE 1456490
- Request Support for write only (set) or read only (get) objects RFE 1459057
- Fixed Superclass reflection issue Issue 1468980
- Fixed Wrong field mapped when an instance is passed in. Issue 1456513
- Fixed No tranformation from a Map attribute to source value Issue 1431086
- Fixed CGLIB Source Class - object not instance of declaring class Issue 1427982

2.0.2 *2006-02-29*

- Added PDF Users Guide
- Request for more log.dubug statements to help diagnosing mapping problems RFE 1445372
- Fixed log.error when an exception is thrown and stop on errors is true RFE 1445376

2.0.1 *2006-02-02*

- Request for allow override to exclude mapping of null src field value RFE 1413480
- Request for allow override to exclude mapping of "" src field value RFE 1417170
- Request check mapping file for duplicate entries RFE 1416905
- Fixed Hint is not used for Array to List Issue 1413429
- Fixed Context Based Mapping and Nested Context based mapping does not work unless a Map Issues 1413443 and 1413451
- Fixed LinkedHashMap not compatible with JDK 1.3 Issues 1419357
- Fixed NPE when sourceFieldValue is null in logFieldMappingError() method

2.0 *2006-01-16*

- Request for create dozer 1.x - 2.x migration documentation. RFE 1397121
- Request for increase performance 4X - 8X. RFE 1381246
- Request for XML tags more consistent. RFE 1381249
- Request for custom create methods. RFE 1386770
- Request for remove static BeanMapper. RFE 1381248
- Fixed exclude-field doesn't work on an indexed property Issue 1404944
- Fixed Better error message for null top level src object Issue 1386663
- Fixed OO domain objs to flat VO obj w/ interface issue Issue 1391777
- Fixed Default ClassMap not getting CustomConverters Issue 1384887

1.5.8.1 *2005-12-08*

- Request for ability to have overloaded set() methods. RFE 1375559
- Request for upgrade to commons-lang-2.1. RFE 1375457

- Request for ability to map field w/out get() set() Methods. This is a partial implementation. RFE 1373285
- Request for if no destination for List, Set, Map inst. source class when appropriate. RFE 1373219
- Request for enhanced ability to copy map to map. RFE 1372011
- Request for Set to Array, Array to Set, Set to List, List to Set. RFE 1370482
- Fixed compatibility with JDK 1.3. At one point we had this…and then reverted back by using LinkedHashSet.
- Fixed Null Value in String Array causes NPE Issue 1373824
- Fixed Deep property mapping and Inheritance Issue 1372000

1.5.8 *2005-11-29*

- Request for enhancement remove dependency on Castor. RFE 1369801
- Request for enhancement ability to Initialize DozerBeanMapper. RFE 1367803
- Request for ability to inject custom converters. RFE 1363361
- Request for enhancement for making CustomConverters global to all mapping files. RFE 1363358
- Fixed InterFace isAssignableFrom Class Issue 1365701
- Fixed Iterator to Array/List Issue 1364373
- Fixed Array to Array mapping Issue 1361105

1.5.7 *2005-11-15*

- Added support to map to/from a Map or Struts **like** DynaActionForm object. Dozer now supports mapping to/from any Map backed property. It can implement the Map Interface or be a custom map object: Forum Thread 452530
- Request for enhancement declare a class as copy-by-reference RFE 1345821

- Request for enhancement No way to reference self (this) in field mapping RFE 1277096
- Request for enhancement for Collapse <field-deep> tag into <field> tag. All <field-deep> tags need to be replaced with <field> tags. RFE 1347953
- Applied patch for Spring bean factory injection: Patch 1349799
- Fixed Configuration Overrides Not Working Issue 1352438
- Fixed multiple levels of custom mapping processed in wrong order Issue 1346370
- Fixed regression on finding most specific destination mapping Issue 1346342
- Fixed ExcludeFieldMap not inherited Issue 1345816

1.5.6 *2005-10-30*

- <field-deep> enhancements: Support for copy-by-reference and type (one-way) attributes.
- Added News section on home page. Latest release number is now on left-hand side navigation column.
- Added support for copying an object by reference: Forum Thread
- Request for enhancement for Custom Bean Creation Factories: RFE 1325412
- Applied patch for MappingProcessor exception handling refactoring: Patch 1333634
- Applied patch for finding most specific mapping: Patch 1325524
- Fixed Empty String value is getting lost during mapping Issue 1342611
- Fixed one-way overriden by default field mappings Issue 1339074
- Fixed Using an Interface for a Custom Converter Class Issue 1342127
- Fixed wrong destination object for list of custom converted object Issue 1332606
- Fixed custom convertor not associated with hints Issue 1321647

1.5.5 *2005-10-13*

- Fixed incorrect set mapping for existing destination set Issue
- Fixed Use of Default ClassMap with date formatting Issue
- If the destination object was mapped and is NULL, we now map it.
- When mapping Array → Array or List → Array do not need hints anymore. This does not apply to Object Arrays.
- Fixed Another Hydrate and Non-Cumulative Issue
- Fixed DTD has field-method and field-iterate methods Issue
- Fixed Field-Deep String to List Error Issue

1.5.4 *2005-09-20*

- Added ability to Map the destination object to Interfaces, Abstract Classes, or Super Classes. See this thread for more information
- Fixed the Hydrate and Non-Cumulative Issue
- Removed samples folder until we can make it more up-to-date.

1.5.3 *2005-09-16*

- Fixed Custom Mapping and made the interface cleaner issue
- Fixed blank string conversion bug 1292080

1.5.2 *2005-09-14*

- Added support for multiple source and destination hints in Collection and Array mapping
- Fixed Mapping inheritance issue(s)
- Iterate type methods can now return an Iterator for their get() method.

1.5.1 *2005-09-07*

- Added <field> level one-way mapping. i.e. <field type="one-way">
- Removed <field-method> and <field-iterate-method> and now use the <field> in combination with attributes. See this thread for more information. This has also been updated in the documentation.
- DTD validation was disabled. It is now working.
- Thanks to Seb :) the ClassLoader issues have been put to bed.

1.5.0.1 *2005-08-31*

- Fixed ClassLoader issue

1.5.0 *2005-08-30*

- Fixed map by reference defect 1264347
- Integrated with Spring. Dozer can be used statically and as a bean
- Fixed a global wildcard bug.
- Upgraded to caster 0.9.6 to fix for allowing whitespace in the XML files
- Addressed adding vs. updating List. We were not truly updating the list

1.4.6 *2005-08-19*

- Addressed adding vs. updating List objects
- Added GenericFieldMap which extends FieldMap so we can have custom behavior on simple field mappings.
- Finally :) fixed ClassLoader bug 1263159
- Dozer now supports overriding getter() and setter() method names
- Addressed Cloneable VS SerializationUtils.clone() issue

1.4.5 *2005-08-18*

- Addressed the ClassLoader defect issue

- Addressed the Pass By Reference issue
- Addressed the field exclude getter() setter() issue

1.4.4 *2005-08-16*

- Added one-way mapping at mapping level into DTD
- Added unit test for one-way mapping
- <field> level tag source and destination hint functionality did not work unless it was an array or list. This has been fixed.

1.4.3 *2005-08-15*

- Moved to DTD validation from XSD validation per defect ID 1256049
- Added <field-exclude> tag.
- Fixed bug where branch and lower level objects which were on the source and already been instantiated were being new()ed up on conversion back to the source.
- If a Java Bean did not have a getter() or setter() method we would throw NPE. We now throw a MappingException with description.

1.4.2 *2005-08-10*

- Support for JDK 1.3

1.4.1 *2005-07-28*

- Deep field output for logging arrays
- Field-deep tag now supports sourceTypeHint and destinationTypeHints

1.4 *2005-07-11*

- Removed test data retrieval from Spring
- MapperException is now a RuntimeException

- Fields in a base class can now be mapped in the mapping file and all of the corresponding sub-classes will not have to specify those fields
- Multiple mapping files can be added to the dozerContext.xml file.

1.3.3 *2005-06-07*

- Added Getting Started section to documentation
- Fixed bug inability to locate mapping file in a jar file at runtime

1.3.2 *2005-06-03*

- Fixed bug 1214291 NullPointerException when no fields
- Changed license to Apache, Version 2.0

1.3.1 *2005-06-02*

- Fixed incorrect error message stating that beanmapping.xml was not in the classpath
- Fixed documentation so that images appear offline
- Added note about using log4j.properties file

1.3 *2005-06-01*

- Fixed deep field mapping logic
- Renamed castormappings.xml to dozercastormappings.xml
- Changed default dozer mappings file to dozerBeanMapping.xml. beanmapping.xml is still supported for backwards compatability.
- Allow override of default mapping file that will get loaded by using system property. -Ddozer.configuration=myBeanMapping.xml
- Resolved some of the pmd errors
- Added a validate method to Mapping Reader. We need to add more.

- Apply top level configuration props to children during initialization process. Removed double checking in the code
- Apply top level configuration props to default class map in processor.
- Added some todos to working doc.
- Added deep mapping examples to samples
- XSD Schema validation
- XSD Schema is now on the website

1.2 *2005-05-03*

- Created a dozer-full.jar file which includes all of the run-time dependency classes needed
- Mappings file can now be empty except for begin and end mappings tag
- Added global configuration XML to contain settings for wildcard, dateformat, error handling, and custom converters
- changes to allow each mapping element override global settings, and some even at the field level
- added a check for infinite loops during mapping
- user can specify whether or not Dozer throws exceptions or eats exceptions
- added the ability to specify custom converter objects to handle special mapping/conversion cases
- created a mapping processor object to allow the use of local variables. Mapper class methods are still static for ease of use.
- Added more unit tests, as always

1.1 *2005-04-06*

- fixed mapping bug with methodMap and methodDehydrate where it would transform but not convert

- many infrastructure changes: moved beanmapping.xml parsing to castor
- fixed bug where bogus data conversion from String to Wrapper would not produce any errors
- The code was changed to get the default mappings after it loads the custom mappings
- String conversions to a Date object based on a date format
- beanmapping.xml file now has field-deep, field-hydrate, field-dehydrate, field-method, and field-iterate-method elements
- Added more unit tests