Dozer Type Conversion

Dozer Type Conversion.

Dozer is a fast and flexible framework for mapping back and forth between Java Beans. Coupled with Camel's automatic type conversion, it's a formidable tool for dealing object to object mapping headaches that crop up in enterprise integration projects.

To explain how Dozer can be uses within Camel we'll use the following example of a simple Customer Support Service. The initial version of the Service defined a 'Customer' object used with a very flat structure.

Legacy Customer Service Class

```
public class Customer {
    private String firstName;
    private String lastName;
    private String street;
    private String zip;

    public Customer() {}

    public Customer(String firstName, String lastName, String zip, String street) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.zip = zip;
        this.street = street;
    }

    public String getFirstName() {
        return firstName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    ... getters and setters for each field
```

In the next version it was decided to structure the data better in the model by moving the address data into its own type, with the resultin domain object ending up looking like

Next Gen Customer object

```
public class Customer {
    private String firstName;
    private String lastName;
    private Address address;

    public Customer() {}

    public Customer(String firstName, String lastName, Address address) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.address = address;
    }
    ....


public class Address {
    private String zipCode;
    private String streetName;

    public Address() {}

    public Address(String zipCode, String streetName) {
        this.zipCode = zipCode;
        this.streetName = streetName;
    }
```

Much nicer! But as often occurs, the previous version of the service, with the old flat 'Customer' object, was in production with a client and the project must support the legacy interface. To support both versions, we must add a mechanism to convert between the old Customer service type and the new Customer domain type and back again. It would be a simple matter to write a custom converter class to map between them, but this may not be the only service/domain inconsistency and these tedious and error prone custom mappings could quickly start to add up, and bugs with them.

To a large extent the two object share identical structure, with only the address representation being different. It would be very helpful if there were a practical way to to automate this kind of mapping, such that the similar properties could get mapped automatically and only the inconsistencies requiring custom mapping.

This is where Dozer comes in; It uses reflection to map data between two bean types using a set of simple mapping rules. Where no rule is specified, dozer will attempt to map between them by using matching properties of two beans. In this way focus can be given to the inconsistencies between the beans i.e. the address properties, knowing that dozer will automatically match and convert the others.

Configuring Dozer

Dozer's configuration is extremely flexible and many mapping scenarios are covered here. For our simple example, the configuration looks like the following.

mapping.xml

```
<mappings xmlns="http://dozer.sourceforge.net" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://dozer.sourceforge.net  http://dozer.sourceforge.net/schema/beanmapping.xsd">
  <mapping>
    <class-a>org.apache.camel.converter.dozer.service.Customer</class-a>
    <class-b>org.apache.camel.converter.dozer.model.Customer</class-b>
    <field>
      <a>street</a>
      <b>address.streetName</b>
    </field>
    <field>
      <a>zip</a>
      <b>address.zipCode</b>
    </field>
  </mapping>
</mappings>
```

Support for Dozer in Camel

Camel provides a simple mechanism to integrate Dozer Mappers with it's own powerful Type Conversion framework. Its configured by creating an instance of `DozerTypeConverterLoader` providing it the camel context and an optional Dozer mapper. If no mapper is supplied, Camel's registry will be searched for suitable instances. The loader will query the Dozer Mapper for the the types it converts and a register them with Camel's type conversion framework to be handled by the mapper.

Limitation

The Camel Dozer type converter does not support having the same type conversion paris in different mapping ids (eg map-id) in Dozer.

In Java it can be configured as follows:

```
DozerBeanMapper mapper = new DozerBeanMapper(Arrays.asList(new String[]{"mapping.xml"}));
new DozerTypeConverterLoader(camelContext, mapper);
```

Or in Spring

```
<!-- the registry will be scanned and 'mapper' below will be found and installed -->
<bean id="dozerConverterLoader" class="org.apache.camel.converter.dozer.DozerTypeConverterLoader" />

<bean id="mapper" class="org.dozer.DozerBeanMapper">
  <property name="mappingFiles">
    <list>
      <value>mapping.xml</value>
    </list>
  </property>
</bean>
```

Configuring in OSGi blueprint

Available as of Camel 2.12

When using Dozer with OSGi Blueprint then its works better by configuring Dozer using the `org.apache.camel.converter.dozer.DozerBeanMapperConfiguration` instead of `org.dozer.DozerBeanMapper`, as shown below:

```
<bean id="dozerConverterLoader" class="org.apache.camel.converter.dozer.DozerTypeConverterLoader">
  <argument index="0" ref="myCamel"/>
  <argument index="1" ref="mapper"/>
</bean>

<bean id="mapper" class="org.apache.camel.converter.dozer.DozerBeanMapperConfiguration">
  <property name="mappingFiles">
    <list>
      <value>mapping.xml</value>
    </list>
  </property>
</bean>

<camelContext id="myCamel" xmlns="http://camel.apache.org/schema/blueprint">
  ...
</camelContext>
```

Now, where necessary, Camel will use Dozer to do conversions; In our case between the new domain and legacy Customer types e.g.

```
// given the following route
from("direct:legacy-service-in").bean(new CustomerProcessor());

// and a processor

public class CustomerProcessor {

    public Customer processCustomer(org.apache.camel.converter.dozer.model.Customer customer) {
       ...
    }
}

// service objects can be sent to the processor and automagically converted by Camel & Dozer
template.sendBody("direct:legacy-service-in",new org.apache.camel.converter.dozer.service.Customer("Bob",
"Roberts", "12345", "1 Main st."));
```