BeanUtils FAQ

The Apache Jakarta BeanUtils Frequently Asked Questions Page

Introduction

This page is intended to gather answers to questions that are regularly asked on the beanutils email lists.

If you discovered something about BeanUtils that you think other people may find useful, please edit this page to add that information.

Why isn't String -> Date conversion provided by default?

Simply because different regions of the world use different date layouts. There isn't any date format that is a reasonable built-in default.

- USA: mm-dd-yyyy, mm/dd/yyyy
- Europe/Pacific: dd-mm-yyyy
- Other popular formats: yyyy-mm-dd, yyyymmdd

If you want BeanUtils to do implicit String->Date conversions for you, then you just need to register a suitable converter for the date formats you expect to encounter in your input.

How can I correctly convert locale-specific input?

If your code is directly calling conversion-related methods on ConvertUtils ConvertUtilsBean to do data conversion, then you can simply change to using LocaleBeanUtils LocaleBeanUtilsBean/LocaleConvertUtils/LocaleConvertUtilsBean instead. The Locale-aware classes will automatically detect the locale of the host machine and set up appropriate converters for that locale. Alternatively you can explicitly create LocaleConvertUtilsBean instances providing a particular locale.

If your code is calling the property-related methods on BeanUtils BeanUtilsBean methods, and you want the automatic type conversion facilities used to be locale-aware then you might want to look at using the equivalent methods on the LocaleBeanUtils or LocaleBeanUtilsBean classes. However because the property-related methods on these classes are not used nearly as often as the property methods on the standard (non-locale-aware) classes, they may not be as well debugged and some features may be missing.

A safer alternative to either of the above is to register custom locale-aware converters with ConvertUtils or ConvertUtilsBean:

```
LongLocaleConverter  longLocaleConverter = new LongLocaleConverter(Locale.GERMAN);
ConvertUtils.register(longLocaleConverter, Long.class);
// now any call to any method on the BeanUtils or ConvertUtils classes which involves
// converting a string to a Long object will use a LongLocaleConverter which is customised
// to handle the German locale.
```

Of course the above will modify the default behaviour across the entire current application (or the current webapp if the code is running in a container environment; see the javadoc for method BeanUtils.getInstance for more information).

If you do not like the idea of changing the ConvertUtils BeanUtils behaviour so widely, then of course you can always create a BeanUtilsBean instance and customise only the behaviour of that instance:

```
ConvertUtilsBean convertUtilsBean = new ConvertUtilsBean();
// here, customise the convertUtilsBean as required by registering custom converters

PropertyUtilsBean propertyUtilsBean = new propertyUtilsBean();
BeanUtilsBean beanUtilsBean = new BeanUtilsBean(convertUtilsBean, propertyUtilsBean);

// now methods on the beanUtilsBean object will use the custom converters registered
// on the associated ConvertUtilsBean instance.
```

How can I customise the conversion?

If you don't like the default way beanutils converts strings to various datatypes, then simply register a custom converter.

So for example if you would like whitespace to be ignored when converting strings to numeric values, then create your own converter classes and register them with ConvertUtils for the datatypes you want to be affected. Note that in this case it would be easier to write a generic "filter" class that wraps the existing converters rather than create new converters classes:

```
private static class WhiteSpaceConverterFilter implements Converter {
  Converter delegate;

  public WhiteSpaceConverterFilter(Converter delegate) {
    this.delegate = delegate;
  }

  public Object convert(Class clazz, Object value) {
    if (value instanceof String) {
      return delegate.convert(clazz, value.toString().trim());
    } else {
      return delegate.convert(clazz, value);
    }
  }
}

ConvertUtils.register(new WhiteSpaceConverterFilter(new IntegerConverter()), Integer.TYPE);
ConvertUtils.register(new WhiteSpaceConverterFilter(new IntegerConverter(), Integer.class);
ConvertUtils.register(new WhiteSpaceConverterFilter(new LongConverter()), Long.TYPE);
ConvertUtils.register(new WhiteSpaceConverterFilter(new LongConverter()), Long.class);
....
```

One particular case of customising conversion is to make the conversion locale-aware. See the FAQ entry titled "How can I correctly convert locale-specific input?" for specific information about this.

How can I customise conversions from type X to String?

Sorry, but ConvertUtils isn't really designed to do that. It is fundamentally about mapping strings (from xml input or web forms, etc) to objects.

The standard converter to the String type simply calls toString on its input object, so you can customise the output for objects of any particular class by modifying the toString method of that class.

Or you could replace the converter for target type String.class with a custom converter which inspects the type of the object and does a big switch statement. Or maybe even look up a mapping table of subconverter objects based on the type of the object being converted.

But in general if you need to do this, then ConvertUtils is probably the wrong tool for your job. You might like to look at morph.sourceforge.net, google for alternatives or craft your own solution.

Why do I get ConversionException when using RowSetDynaClass with Oracle?

Oracle's JDBC driver is broken. When a query is run against a table containing a Timestamp column, the result set's metadata reports the column type (correctly) as javax.sql.Timestamp. And therefore the DynaClass object created to represent the query has a field of type javax.sql.Timestamp to store the data for that column in. However calling resultset.getObject on that column returns an object of type javax.sql.Date⚠️, which then of course cannot be assigned to the corresponding field on a DynaBean object with that DynaClass.

This has been reported with Oracle10g, driver ojdbc14 version 10.1.0.4.

Possible solutions (untested) are:

- subclass RowSetDynaClass and override the copy method to fudge the conversion.
- subclass BasicDynaBean and override the set method to handle the problem, then subclass RowSetDynaClass and override createDynaBean to return instances of your custom bean that handles broken Oracle resultsets.

If you have a license for Oracle, don't forget to send in a complaint!

Do you know that the version number in the MANIFEST.MF for release 1.7 is wrong?

Yes, we know. It says 1.6. So does release 1.6.1.

Now the build system has moved to Maven that won't happen again.

Why do I get ClassCastException when accessing nested maps?

Say you have a map which contains a nested map ala:
nestedmap.put ("key2", "somevalue"); map.put ("key1", nestedmap);

If you use the following syntax:
String val = PropertyUtils.getProperty(bean, "map(key1)(key2)");

It will actually return "map(key1)", which will be a Map, not a String.

BeanUtils uses '.' as a property separator, and the second property should actually used mapped syntax, since it is directly accessing a map now, thus the correct usage should be:
String val = PropertyUtils.getProperty(bean, "map(key1).key2");