

# JDBC Types

by Thomas Mahler, Armin Waibel

## Table of contents

1 Mapping of JDBC Types to Java Types.....	2
2 Type and Value Conversions - The FieldConversion Interface.....	3
2.1 Introduction.....	3
2.2 The problem.....	3
2.3 The Solution.....	3

## 1. Mapping of JDBC Types to Java Types

OJB implements the mapping conversions for JDBC and Java types as specified by the JDBC 3.0 specification (see JDBC 3.0 specification *Appendix B, Data Type Conversion Tables*). See the [table below](#) for details.

If a sql-java type mapping is needed, that doesn't match the java types defined by the specification, e.g. a field in the persistent object class is of type `int[]` and the DB type is `VARCHAR` or a `List` field have to be mapped to `VARCHAR` a [field-conversion](#) class can be used.

JDBC Type	Java Type
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
BOOLEAN	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp
CLOB	Clob
BLOB	Blob
ARRAY	Array
DISTINCT	mapping of underlying type
STRUCT	Struct
REF	Ref
DATALINK	java.net.URL

JAVA_OBJECT	underlying Java class
-------------	-----------------------

## 2. Type and Value Conversions - The FieldConversion Interface

### 2.1. Introduction

A typical problem with O/R tools is mismatching datatypes: a class from the domain model has an attribute of type *boolean* but the corresponding database table stores this attribute in a column of type *BIT* or *int*.

This example explains how OJB allows you to define **FieldConversions** that do the proper translation of types and values.

The source code of this example is included in the OJB source distribution and resides in the test package `org.apache.ojb.broker`.

### 2.2. The problem

The test class `org.apache.ojb.broker.Article` contains an attribute `isSelloutArticle` of type `boolean`:

```
public class Article implements InterfaceArticle
{
    protected int articleId;
    protected String articleName;

    // maps to db-column Auslaufartikel of type int
    protected boolean isSelloutArticle;

    ...
}
```

The corresponding table uses an `int` column (`Auslaufartikel`) to store this attribute:

```
CREATE TABLE Artikel (
    Artikel_Nr          INT PRIMARY KEY,
    Artikelname         CHAR(60),
    Lieferanten_Nr      INT,
    Kategorie_Nr        INT,
    Liefereinheit       CHAR(30),
    Einzelpreis         DECIMAL,
    Lagerbestand        INT,
    BestellteEinheiten  INT,
    MindestBestand      INT,
    Auslaufartikel      INT
)
```

### 2.3. The Solution

OJB allows to use predefined (or self-written) **FieldConversions** that do the appropriate mapping. The **FieldConversion** interface declares two methods: `javaToSql(...)` and `sqlToJava(...)`:

```
/**
 * FieldConversion declares a protocol for type and value
 * conversions between persistent classes attributes and the columns
 * of the RDBMS.
 * The default implementation does not modify its input.
 * OJB users can use predefined implementation and can also
 * build their own conversions that perform arbitrary mappings.
 * the mapping has to be defined in the xml repository
 * in the field-descriptor.
```

```

*
* @author Thomas Mahler
*/
public interface FieldConversion extends Serializable
{
    /**
     * convert a Java object to its SQL
     * pendant, used for insert & update
     */
    public abstract Object javaToSql(Object source) throws ConversionException;

    /**
     * convert a SQL value to a Java Object, used for SELECT
     */
    public abstract Object sqlToJava(Object source) throws ConversionException;
}

```

The method `FieldConversion.sqlToJava()` is a callback that is called within the OJB broker when Object attributes are read in from JDBC result sets. If OJB detects that a `FieldConversion` is declared for a persistent classes attributes, it uses the `FieldConversion` to do the marshalling of this attribute.

For the above mentioned problem of mapping an int column to a boolean attribute we can use the predefined `FieldConversion Boolean2IntFieldConversion`. Have a look at the code to see how it works:

```

public class Boolean2IntFieldConversion implements FieldConversion
{
    private static Integer I_TRUE = new Integer(1);
    private static Integer I_FALSE = new Integer(0);

    private static Boolean B_TRUE = new Boolean(true);
    private static Boolean B_FALSE = new Boolean(false);

    /**
     * @see FieldConversion#javaToSql(Object)
     */
    public Object javaToSql(Object source)
    {
        if (source instanceof Boolean)
        {
            if (source.equals(B_TRUE))
            {
                return I_TRUE;
            }
            else
            {
                return I_FALSE;
            }
        }
        else
        {
            return source;
        }
    }

    /**
     * @see FieldConversion#sqlToJava(Object)
     */
    public Object sqlToJava(Object source)
    {
        if (source instanceof Integer)
        {
            if (source.equals(I_TRUE))
            {
                return B_TRUE;
            }
            else
            {

```

```

        return B_FALSE;
    }
    else
    {
        return source;
    }
}

```

There are other helpful standard conversions defined in the package `org.apache.ojb.broker.accesslayer.conversions`: Of course it is possible to map between `java.sql.date` and `java.util.date` by using a `Conversion`. A very interesting `Conversion` is the `Object2ByteArrayFieldConversion` it allows to store inlined objects in `varchar` columns!

Coming back to our example, there is only one thing left to do: we must tell OJB to use the proper *FieldConversion* for the *Article* class. This is done in the XML repository file. The [field-descriptor](#) allows to define a *conversion* attribute declaring the fully qualified *FieldConversion* class:

```

<!-- Definitions for test.ojb.broker.Article -->
<class-descriptor
  class="org.apache.ojb.broker.Article"
  proxy="dynamic"
  table="Artikel"
>
  ...
  <field-descriptor
    name="isSelloutArticle"
    column="Auslaufartikel"
    jdbc-type="INTEGER"
    conversion="org.apache.ojb.broker.accesslayer.conversions.Boolean2IntFieldConversion"
  />
  ...
</class-descriptor>

```