org.jax.mgi.shr.types Design Document

Author: Mike Walker

Created: March 19, 2003

Last Modified: May 21, 2004 13:09

1 Purpose of Document

This document describes the classes belonging to the org.jax.mgi.shr.types package and provides source code examples for common usage patterns.

2 Introduction

The org.jax.mgi.shr.types package is comprised of the classes which handle or operate on a set of known java data types. Applications can be simplified by deciding up front which types are appropriate for use, especially when the application is expected to provide database access and commonly does conversion between java types and JDBC types, or provides file access and commonly does conversion between Strings and other java types. The java types handled by this package are as follows.

- int
- float
- boolean
- Integer
- Float
- Boolean
- String
- Timestamp

This list influences the allowed range of JDBC types which are handled by the **SQLDataManager** class (see the org.jax.mgi.shr.dbutils package). Furthermore, this list influences the range of types handled when validation occurs during the bcp process inside the **BCPManager** class (see the org.jax.mgi.shr.dbutils.bcp package).

This product integrates with the org.jax.mgi.shr.exception package.

This product was coded and tested on Java 1.4 JDK.

3 Overview of Classes

The **Converter** class is a class which provides static conversion methods for converting a predefined set of data types to and from Strings. Internally, this class uses conversion methods pro-

vided in Java. But whereas Java throws a runtime exception for conversion errors, this class handles java runtime exceptions and instead throws them as compile time exceptions (meaning you would have to handle the conversion exceptions or declare them in your throw clause). There are also methods for converting primitive types to their corresponding wrapper classes. These methods are used internally by the **DataVector** class discussed next. See Example 1 for usage.

EXAMPLE 1. Using a Converter

```
int seqID;
Timestamp millennium;
try
{
  seqID = Converter.toPrimitiveInt('123');
  millennium = Converter.toTimestamp('2000/01/01');
}
catch (DataException)
{
   // handle the conversion exception
}
```

The **DataVector** class is an extension of the Vector class. It adds methods for setting, adding and removing primitive values to the Vector. Since the Vector class only accepts instances of classes, primitives need to be promoted to their corresponding wrapper classes before they can be stored in the Vector. See Example 2. A **DataVector** object can be used anywhere a Vector is required. Note that since these primitives are not being stored as primitives, when you call the accessor methods on the **DataVector** you will get back the wrapper classes instead.

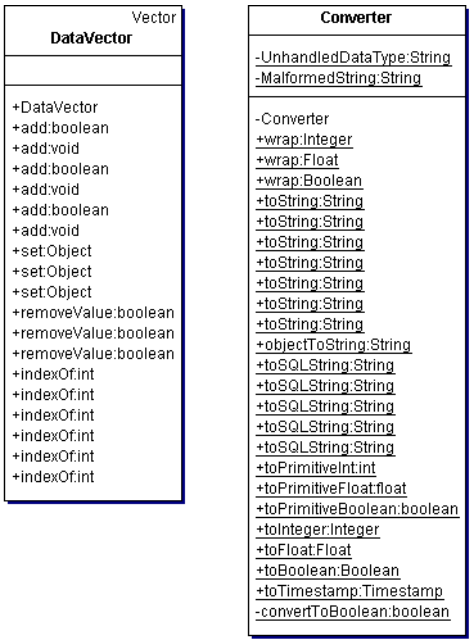
EXAMPLE 2. Using a DataVector

```
DataVector v = new DataVector();

int seqID = 1234;
float ageMin = 0.0;
float ageMax = 21.0;
boolean new = true;

v.add(seqID);
v.add(new);
v.add(ageMin);
v.add(ageMax);
```

4 Class Diagram



5 Type Conversion Details

The **Converter** class will convert to and from Strings those data types listed in Section 2. This section details some of the format restrictions imposed on these conversions.

When converting boolean values to Strings they become the values "1" or "0". This was done to make conversion compatible with the interpretation for boolean strings imposed by Sybase. The following String values are acceptable when converting to boolean or Boolean: ‘true’, ‘false’, ‘1’, ‘0’, ‘yes’ or ‘no’. There is no regard for case.

String values for the numeric types would have to agree with their corresponding numeric formats. Consequently, String values with decimal points cannot be converted to int or Integer types.

Timestamp conversion expects strings of the following format (with or without the time component):

- yyyy-mm-dd hh:mm:ss.fffffffff
- yyyy/mm/dd hh:mm:ss.fffffffff