WorkingWithColumnConverters

WorkingWithColumnConverters

Sometimes it is desireable to modify a column value retreived from a database column **before** it is stored into a DataObject property. One example might be a database column with possible values '0' or '1' but the desired DataObject property is 'false' or 'true'. Another example might be a String property that should be obfuscated or encryted before it is stored into a column.

The RDB DAS provides a simple "converter" framework that can be used for these cases and others. To use this framework the application developer must provide a new Converter class to provide the desired transformations from column to property and from property back to column. The provided converter must implement the following interface:

```
package org.apache.tuscany.das.rdb;

/**
 * A lightweight Table-column <--> DataObject-property converter framework. Converters allow a user to
 * insert a transformation between a column value and is destination DataObject property value. For example,
 * by default, a VARCHAR column will be represented as a     String in its corresponding DataObject property.
 * A user could insert a converter that transforms the the VARCHAR value to an Integer. If this is done then
 * although the column returns character data, the DataObject property will be an Integer
 */
public interface Converter {

    /**
     * Transform the columnData object to a new value and possibly new type. This should be the inverse
     * operation of #getColumnValue
     *
     * @param columnData
     *            The column value to transorm
     * @return Returns the transformed value
     */
    Object getPropertyValue(Object columnData);

    /**
     * Transform the columnData object to a new value and possibly new type. This should be the inverse
     * operation of #getPropertyValue
     *
     * @param propertyData
     *            The property value to transform
     * @return Returns the transformed value
     */
    Object getColumnValue(Object propertyData);

}
```

The following example illustrates the use of the converter framework to obfuscate a String value before it is stored to the database and deobfuscate it on read.

```
DAS das = DAS.FACTORY.createDAS(getConfig("CustomerConfig.xml"), getConnection());

        // Create and initialize command to read customers
        Command read = das.getCommand("getFirstCustomer");

        // Read
        DataObject root = read.executeQuery();

        //Modify
        root.setString("CUSTOMER[1]/LASTNAME", "Some new name");

        das.applyChanges(root);
```

This example just shows a simple read of a customer DataObject followed setting the last name to some new value. You will have to take my word for it that the value is not written as-is but is instead obfuscated. The actual value written to the database table column is "Fbzr arj anzr" based on the ROT13 algorithm used in the converter.

Converters are specified in the configuration, usually as part of a config.xml file. Here is the config file for the example:

```
<Config xsi:noNamespaceSchemaLocation="http:///org.apache.tuscany.das.rdb/config.xsd" xmlns:xsi="http://www.w3.
org/2001/XMLSchema-instance">

  <Command name="getFirstCustomer" SQL="Select * from CUSTOMER where ID = 1" kind="Select"/>

  <Table tableName="CUSTOMER">
        <Column columnName="ID" primaryKey="true"/>
    <Column columnName="LASTNAME" converterClassName="org.apache.tuscany.das.rdb.test.mappings.
StringObfuscationConverter"/>
  </Table>

</Config>
```

Here is the user-defined converter class:

```
public class StringObfuscationConverter implements Converter {

    public StringObfuscationConverter() {
        super();
    }

    public Object getPropertyValue(Object columnData) {
        return toRot13((String) columnData);
    }

    public Object getColumnValue(Object propertyData) {
        return toRot13((String) propertyData);
    }

    // Utilities

    // A simple, reversible, obfuscation algorithm using a ROT13 implementation
    private String toRot13(String original) {

        int abyte = 0;
        byte[] buffer = {};
        try {
            buffer = original.getBytes("ISO-8859-1");
        } catch (UnsupportedEncodingException e) {
            throw new Error(e);
        }

        for (int i = 0; i < buffer.length; i++) {
            abyte = buffer[i];
            int cap = abyte & 32;
            abyte &= ~cap;
            abyte = ((abyte >= 'A') && (abyte <= 'Z') ? ((abyte - 'A' + 13) % 26 + 'A') : abyte) | cap;
            buffer[i] = (byte) abyte;
        }
        try {
            return new String(buffer, "ISO-8859-1");
        } catch (UnsupportedEncodingException e) {
            throw new Error(e);
        }
    }
}
```