Reading and Writing Binary Data With Java Transformations in PowerCenter

Abstract

The BinaryReader Java transformation reads binary data from a file. The BinaryWriter Java transformation writes binary data as a file. This article describes how to use the Java transformations to read from a file and write binary data into a target.

Supported Versions

- PowerCenter 9.0, 9.1.0

Table of Contents

Overview

You can use a Java transformation to read binary data from a file or write binary data into a target. To use a BinaryReader or BinaryWriter Java transformation, you can download and import the transformations into the repository.

You can use BinaryReader Java transformation to read binary data from sources such as image file, PDF, audio file, or a video file. For example, you copy an image file on a disk to a database BLOB column with the BinaryReader Java transformation.

You can use BinaryWriter Java transformation to write binary data as a flat file. With the BinaryWriter Java transformation, you can simulate behavior similar to the Save As option in Outlook. You can write the attachments from binary sources into a folder with the BinaryWriter Java transformation.

Importing the Java Transformations into the Repository

You can download the Java transformations from the xml files for the BinaryReader and Binary Writer Java transformations from the following location:

https://communities.informatica.com/docs/DOC-5303

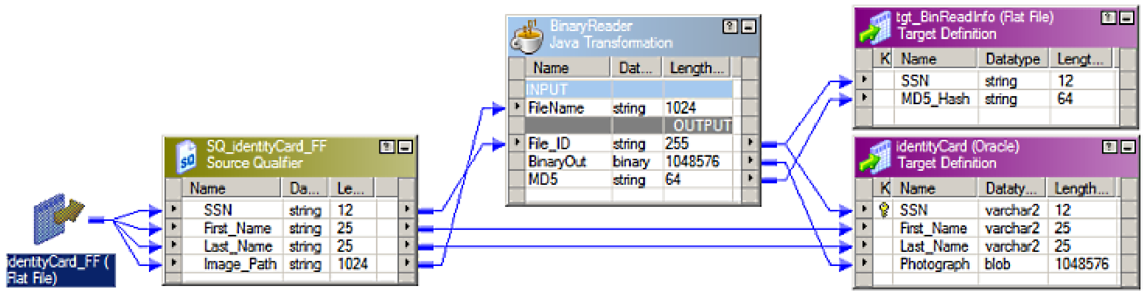
After you download the files, save the files in your local storage. Import the Java transformations into the repository using the PowerCenter Repository Manager or the Designer. Use the transformations in the mapping to read or write binary data.

BinaryReader Java Transformation Example

The BinaryReader Java transformation is a passive, connected transformation that reads any binary file as source data. The BinaryReader Java transformation requires the absolute path of the binary file as input.

You create a mapping to write the JPEG images to the BLOB column in the Oracle target. You create a mapping with a flat file source, the BinaryReader Java transformation, an Oracle target, and a flat file target.

The following figure shows the mapping:



Source File

You have the following source data:

SSN	First_Name	Last_Name	Image_Path
987-65-4320	Lloyd	Warren	E:\ID_Pic01.jpg
987-65-4321	Miriam	Ruiz	E:\ID_Pic02.jpg
987-65-4322	Brad	Moore	E:\ID_Pic03.jpg
987-65-4323	Kyle	Henderson	E:\ID_Pic04.jpg

The Image_Path column contains the binary file path.

The following table describes the properties for the flat file source definition:

Name	Datatype	Length
SSN	string	12
First_Name	string	25
Last_Name	string	25
Image_Path	string	1024

BinaryReader Java Transformation

You configure the BinaryReader Java transformation as a passive transformation and add the ports on the Ports tab when you manually create the transformation.

The following table describes the properties for the BinaryReader Java transformation:

Name	Port	Datatype	Precision	Description
File_Name	Input	string	1024	Receives input from the source Image_Path that has the path of the file to read.
File_ID	Input and Output	string	255	File ID is any unique value associated with the file.
Binary_Out	Output	binary	1048576	Outputs the binary data from the file.
MD5	Output	string	64	The MD5 hash of the file when the file is successfully read.

Target Files

The flat file target definition contains ports for SSN and MD5 hash.

The following table describes the Oracle target definition:

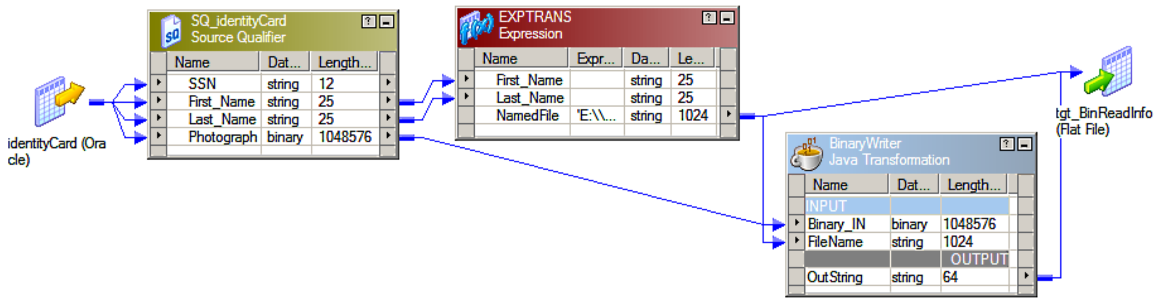
Name	Datatype	Length
SSN	varchar2	12
First Name	varchar2	25
Last Name	varchar2	25
Photograph	blob	1048576

BinaryWriter Java Transformation Example

The BinaryWriter Java transformation is a passive, unconnected transformation that you can use to write binary data as a file. For example, you have an email with an attachment. You can select all the attachments from the source to write into a disk with the BinaryWriter Java transformation.

You want to read images from an Oracle table with the BLOB entry and write them to a target directory.

The following mapping shows how the BinaryWriter Java transformation writes binary data from the source to the disk:



Source File

You have the following data from an Oracle source:

SSN	First_Name	Last_Name	Photograph
987-65-4320	Lloyd	Warren	<binary data>
987-65-4321	Miriam	Ruiz	<binary data>
987-65-4322	Brad	Moore	<binary data>
987-65-4323	Kyle	Henderson	<binary data>

The photograph column contains the binary data.

You can import the Oracle table with the following properties in an Oracle source definition:

Name	Datatype	Length	Key
SSN	varchar2	12	Primary Key
First_Name	varchar2	25	Not a Primary key
Last_Name	varchar2	25	Not a Primary key
Photograph	blob	1048576	Not a Primary key

Expression Transformation

You configure the Expression transformation to change the file names in the following format:

```
<First_Name>_<Last_Name>.jpg
```

You configure the following expression in the Expression transformation:

```
'E:\\card_images\\' || First_Name || '_' || Last_Name || '.jpg'
```

BinaryWriter Java Transformation

The BinaryWriter Java transformation requires the absolute path in the File_Name port.

You configure the BinaryWriter Java transformation with the following properties:

Name	Input/Output	Datatype	Precision	Description
File_Name	Input	string	1024	The file name including the absolute path of the file to create.
Binary_In	Input	binary	1048576	The binary data from the source transforms it into a file on the file system.
OutString	Output	string	64	The MD5 hash of the file when the file is written to the target.

Target File

The target file contains the SSN and the MD5 output. You can view the binary data as file in the location specified against the File_Name port of the BinaryWriter Java transformation.

Java Source Code for the Java Transformations

The following four Java code snippets define the transformations logic. The appropriate code entry tabs of the Java transformations define the BinaryReader and BinaryWriter Java transformations.

Code in the BinaryReader Transformation

The following code entry tabs contain the Java code snippets that define the BinaryReader Java transformation:

Import Packages Tab

```
import java.io.InputStream;
import java.io.File;
import java.io.FileInputStream;
import java.math.BigInteger;
import java.security.MessageDigest;
```

Helper Code Tab

```
static int countNullRows;
static Object lock = new Object();
static java.io.File fileIn = null;
static InputStream is = null;
static MessageDigest digest = null;
```

On Input Row Tab

```
if(!isNull("FileName")){
   try {
          fileIn = new java.io.File(FileName);
          is = new FileInputStream(fileIn);
          digest = MessageDigest.getInstance("MD5");
   } catch (Exception e) {
          logError("Unable to create open the file " + FileName);
   }
   long length = fileIn.length();
   if (length > Integer.MAX_VALUE) {
          logError("File is too large");
   }
   byte[] bytes = new byte[(int) length];

   int offset = 0;
   int numRead = 0;
   try {
          while (offset < bytes.length && (numRead = is.read(bytes,
offset, Math.min(bytes.length - offset, 512 * 1024))) >= 0) {
                 digest.update(bytes, 0, numRead);
                 offset += numRead;
          }
   } catch (Exception ex) {
          logError("Error reading file");
   } finally {
          try {
                 is.close();
          } catch (Exception ex) {
                 logError("Error closing file");
          }
   }

   if (offset < bytes.length) {
          bytes = null;
          setNull(MD5);
          setNull(File_ID);
          logError("Could not completely read file " +
fileIn.getName());
   } else {
          byte[] md5sum = digest.digest();
          BigInteger bigInt = new BigInteger(1, md5sum);

          BinaryOut = bytes;
          MD5 = bigInt.toString(16);
          logInfo("Successfully read file " + fileIn.getName() + ",
ID: " + File_ID + ", of size " + bytes.length + " bytes, with MD5
checksum: \"" + MD5 + "\"");
   }
}else{
   synchronized(lock) {
          countNullRows++;
   }
}
```

On End of Data Tab

```
synchronized(lock)
```

```
{
   logInfo("The total number of null rows across partitions is : " +
countNullRows);
}
logInfo("End of data");
```

Code in the BinaryWriter Transformation

The following code entry tabs contain the Java code snippets that define the BinaryWriter Java transformation:

Import Packages Tab

```
import java.io.InputStream;
import java.io.ByteArrayInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.*;
```

Helper Code Tab

```
static int countNullRows;
static Object lock = new Object();
FileOutputStream out = null;
ByteArrayInputStream in = null;
MessageDigest md5;
byte[] digest = new byte[] {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
String fileMD5 = "";
```

On Input Row Tab

```
if(!isNull("Binary_IN") && !isNull("FileName")) {
        try {
         String filename = FileName;

            in = new ByteArrayInputStream(Binary_IN);
            out = new FileOutputStream(filename);
            int c;
            while ((c = in.read()) != -1) {
                out.write(c);
            }
            fileMD5="";
            try {
                md5 = MessageDigest.getInstance("MD5");
                md5.update(Binary_IN);
                digest = md5.digest();
                for (int i=0; i < digest.length; i++) {
                    fileMD5 += Integer.toString( ( digest[i] & 0xff )
+ 0x100, 16).substring( 1 );
                }
            } catch (Exception e) {
                logError(e.toString());
                fileMD5 = "";
            }
            OutString = fileMD5;
            logInfo("Wrote \"" + filename + "\" successfully. MD5: "
+ fileMD5);
        } catch(NullPointerException npex){
            OutString = "";
            logError("The input binary is null " + npex.toString());
        } catch (IOException ioex) {
            OutString = "";
            logError("An I/O error has occured. Check if the target
directory exists and has write previleges." + ioex.toString());
```

```
                } catch(Exception ex){
                    OutString = "";
                    logError("Failed to write row" + ex.toString());
                }finally {
                    if (in != null) {
                        try{in.close();}catch(Exception ex){}
                    }
                    if (out != null) {
                        try{out.close();}catch(Exception ex){}
                    }
                }
            } else {
                    synchronized(lock)
                    {
                          countNullRows++;
                    }
            }
```

On End of Data Tab

```
synchronized(lock)
{
   logInfo("The total number of null rows across partitions is : " +
countNullRows);
}
logInfo("End of data");
```

Author

Goutham Nanjundaswamy
Software QA Engineer