The TMG to HRE database converter

By Nils Tolleshaug, nils@tolleshaug.net

2020-05-22	Nils Tolleshaug	Original version referring to TMG functions.
2020-11-10	Nils Tolleshaug	Full revision and code aligned with HRE Build 25
2020-12-29	Nils Tolleshaug	Update after populating tables in PeopleViewPoint
2021-01-03	Don Ferguson	Minor revisions for accuracy

Content

1. Introduction

The TMG database stores only data for Genealogy projects. The HRE database has a wider scope and allows the user to store data for historical projects where genealogy data in TMG can be considered as a subset in the scope of the HRE database. Based on years of experience with the TMG database, the design of HRE avoids the basic design restrictions in TMG. Typical is that an event in TMG can only have two Principals (husband/wife or two parents etc.) while HRE can have one to many Principals (called Key Associates) for any event. Another observation is that the TMG database structure has evolved over more than 10 years until the year 2014 when the TMG development ceased. Basically, there are compromises and restrictions in the TMG database that do not exist in the HRE database.

The idea behind the converter is to setup Java object structures for both the TMG database and the HRE database and give the programmer a set of methods for accessing data elements of the databases. The programmer can setup a series of statements that extract a data element from the TMG tables and store the data element in the corresponding field in the HRE database tables.

The TMG to HRE conversion is a complex process and therefore the conversion process is split into several passes where each pass converts a certain subset of the data. A typical pass is conversion of Names from TMG to HRE while the next pass can be converting Event data. The passes correspond briefly to the phases in the development plan for the HRE system. The final number of, and data type conversion in each pass is not yet decided.

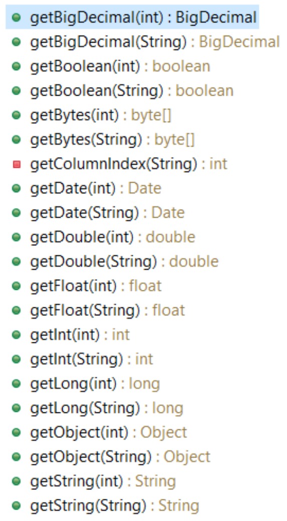
> *Note that the TMG HRE converter is designed as a library to be used for TMG to HRE database conversion and the current implementation is only a testbed that needs to be supplied with the actual conversion statements in each phase.*

In this document the HRE US Sample project is used so the person file is named: **sample_$.dbf**

<u>It is recommended to read this document while inspecting the TMG to HRE converter code.</u>

2. Reading TMG database files

The TMG database is a FoxPro database using the dbf-style of table structures. It has a number of tables for each project named with letters, for example sample_$.dbf, sample_D.dbf, sample_G.dbf etc. The full descriptions of the TMG database tables are found in the document “TMG File Structure”[1] . Each table can have three sub tables: the index table(.cdx), the database(.dbf) and the memo file(.fpt). Memo field data is stored in the memo file. The memo fields are an open-ended String of text referred to by the M field in the TMG database. Please note that only tables that have memo fields (M type) have a corresponding memo file (.fpt).



```
getBigDecimal(int) : BigDecimal
getBigDecimal(String) : BigDecimal
getBoolean(int) : boolean
getBoolean(String) : boolean
getBytes(int) : byte[]
getBytes(String) : byte[]
getColumnIndex(String) : int
getDate(int) : Date
getDate(String) : Date
getDouble(int) : double
getDouble(String) : double
getFloat(int) : float
getFloat(String) : float
getInt(int) : int
getInt(String) : int
getLong(int) : long
getLong(String) : long
getObject(int) : Object
getObject(String) : Object
getString(int) : String
getString(String) : String
```

The Java tool used to read the TMG tables is the library javadbf [2]developed by Alberto Fernández. The javadbf library opens a TMG database file as DBFReader from a Java read stream and reads record by record into DBFRow objects. The content of the DBFRow object is read with method calls corresponding to the datatype (Integer, char, long memo etc) and with a parameter that is either the index of the field in the row or the column heading name (see figure left).

If DBFReader during setup is associated with the corresponding memo file, the memo fields are read as a String field.

The content of the TMG table can be presented on a PC screen with the “DBF Commander” tool - this is a commercial product with a free version with limited functionality. DBF Commander can also present the contents of the table including contents of the memo file.

[1] The Master Genealogist (TMG) File Structures for v9. Last updated July 2014

[2] Library com.linuxense.javadbf , Java library for reading and writing Xbase (dBase/DBF) files https://github.com/albfernandez/javadbf, albfernandez/javadbf is licensed under the GNU Lesser General Public License v3.0 and written by Alberto Fernández

3. Handling SQL database tables with ResultSet

The SQL statements that read data from a database query return the data in a ResultSet. The SELECT statement is the standard way to select rows from a database and view them in a ResultSet. The ResultSet interface represents the result set of a database query. A ResultSet object stores the cursor value that points to the current row in the result set. By referring to the row and column in a ResultSet object the content of a database object can be read from and written to. The methods in the ResultSet interface can be divided into three categories:

Navigational methods:
Used to move the cursor around.

Get methods:
Used to view the data in the columns of the current row being pointed to by the cursor.

Update methods:
Used to update the data in the columns of the current row.
The updates can then be updated in the underlying database as well.

If you do not specify any ResultSet type, you will automatically get one that is TYPE_FORWARD_ONLY. If the ResultSet is set to TYPE_SCROLL_INSENSITIVE the cursor can scroll forward and backward, and the result set is not sensitive to changes made by others to the database that occur after the result set was created.

Navigating a Result Set. There are several methods in the ResultSet interface that involve moving the cursor, including:

1. public void beforeFirst() throws SQLException
   Moves the cursor just before the first row

2 public void afterLast() throws SQLException
  Moves the cursor just after the last row

3 public boolean first() throws SQLException
  Moves the cursor to the first row

4 public void last() throws SQLException
  Moves the cursor to the last row

5 public boolean absolute(int row) throws SQLException
  Moves the cursor to the specified row

6 public boolean relative(int row) throws SQLException
  Moves the cursor the given number of rows forward or backward, from where it is currently pointing

7 public boolean previous() throws SQLException
  Moves the cursor to the previous row. This method returns false if the previous row is off the result set

8 public boolean next() throws SQLException
  Moves the cursor to the next row. This method returns false if there are no more rows in the result set

9 public int getRow() throws SQLException
  Returns the row number that the cursor is pointing to

10 public void moveToInsertRow() throws SQLException
   Moves the cursor to a special row in the result set that can be used to insert a new row into the database. The current cursor location is remembered

11 public void moveToCurrentRow() throws SQLException
Moves the cursor back to the current row if the cursor is currently at the insert row; otherwise, this method does nothing.

The ResultSet interface contains a number of methods for getting data types from the current row. There is a get method for each of the possible data types, and each get method has two versions − one that uses column name as the input parameter and the other that uses the column index.

The ResultSet interface contains a collection of update methods for updating the data of a result set. As with the get methods, there are two update methods for each data type. Like the get methods the update methods use either column name or column index to identify the field in a row.

Updating a row in the result set changes the columns of the current row in the ResultSet object, but not in the underlying database. To update your changes to the row in the database, you need to invoke one of the following methods:

1 public void updateRow()
Updates the current row by updating the corresponding row in the database

2 public void deleteRow()
Deletes the current row from the database

3 public void refreshRow()
Refreshes the data in the result set to reflect any recent changes in the database

4 public void cancelRowUpdates()
Cancels any updates made on the current row

5 public void insertRow()
Inserts a row into the database.
This method can only be invoked when the cursor is pointing to the insert row.

4. The TMG to HRE classes

This is a summary of the classes in the TMG-HRE converter.

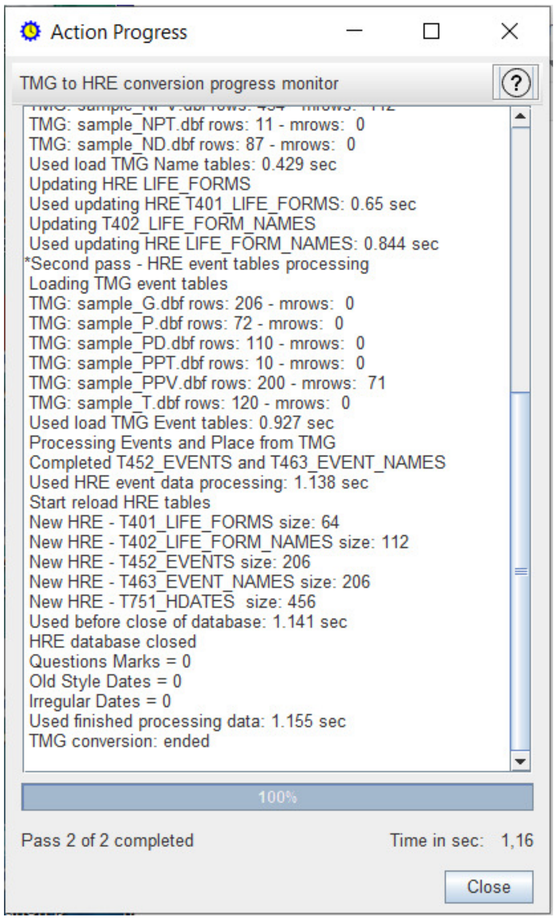
class TMGHREconverter extends SwingWorker<String, String>

This class activates the TMGfolderChooser to select the TMG project's PJC file, reads the content of the PJC file and presents project data in TMGHREprogressMonitor. The .pjc file contains the TMG project data such as the TMG database version, the database owner, the last data for update etc. This class is the main class for the start of the converter and at first initiation loads the TMG and HRE database tables. The class initiates the phases in the conversion process, controls the conversion process and updates status of the conversion process via the Progress Monitor. The first version of the TMG to HRE converter has implemented two passes:

Pass 1 – Person and name data extracted from TMG tables:
    sample_$.dbf, sample_N.dbf, sample_NPV.dbf, sample_NPT.dbf and sample_ND.dbf
Pass 2 – Event and place data from TMG tables;
    sample_G.dbf, sample_PPV.dbf, sample_PPT.dbf, sample_PD.dbf

class TMGHREprogressMonitor extends JFrame implements PropertyChangeListener

This displays a window with text monitor output to a text area to present status updates from TMGHREconverter.



```
TMG: sample_NPT.dbf rows: 11 - mrows:  0
TMG: sample_ND.dbf rows: 87 - mrows:  0
Used load TMG Name tables: 0.429 sec
Updating HRE LIFE_FORMS
Used updating HRE T401_LIFE_FORMS: 0.65 sec
Updating T402_LIFE_FORM_NAMES
Used updating HRE LIFE_FORM_NAMES: 0.844 sec
*Second pass - HRE event tables processing
Loading TMG event tables
TMG: sample_G.dbf rows: 206 - mrows:  0
TMG: sample_P.dbf rows: 72 - mrows:  0
TMG: sample_PD.dbf rows: 110 - mrows:  0
TMG: sample_PPT.dbf rows: 10 - mrows:  0
TMG: sample_PPV.dbf rows: 200 - mrows:  71
TMG: sample_T.dbf rows: 120 - mrows:  0
Used load TMG Event tables: 0.927 sec
Processing Events and Place from TMG
Completed T452_EVENTS and T463_EVENT_NAMES
Used HRE event data processing: 1.138 sec
Start reload HRE tables
New HRE - T401_LIFE_FORMS size: 64
New HRE - T402_LIFE_FORM_NAMES size: 112
New HRE - T452_EVENTS size: 206
New HRE - T463_EVENT_NAMES size: 206
New HRE - T751_HDATES  size: 456
Used before close of database: 1.141 sec
HRE database closed
Questions Marks = 0
Old Style Dates = 0
Irregular Dates = 0
Used finished processing data: 1.155 sec
TMG conversion: ended
```

Pass 2 of 2 completed   Time in sec: 1,16

class TMGfolderChooser extends JDialog

With folder chooser the user first selects the project folder for the TMG project, opens the folder and selects the TMG project's PJC file. A selection of relevant PJC file data is presented in the TMGHREprogressMonitor window.

class TMGloader

Loads the TMG tables into the system to make the tables accessible in the conversion process. Uses the class TMGtypes to set readable names to the TMG classes. Uses TMGtableData to hold the data for each TMG table (see the next section of this document).

class TMGtypes

Variables for naming of TMG tables are:

```
String PERSON = "$";
String SOURCE_TYPE = "A";
String FOCUS_GROUP_MEMBER = "B";
String FLAG = "C";
String DATA_SET = "D";
String DNA = "DNA";
String EVENT_WITNESS = "E";
String EVENT = "G";
```

class TMGtableData

This class stores the data read from the TMG tables and has one instance of the class for each TMG table. The DBFRow object containing data for each row in the TMG table is created from the TMG table and referred to by Java pointer ArrayList.

ArrayList<DBFRow> PIDlist = new ArrayList<DBFRow>();

so that DBFRow object can be indexed through the ArrayList.

The tables in TMG database rows can have multiple primary indexes (PID). For example, a person with PID=3 can refer to 4 rows in the name table as the person has 4 names recorded in the database. To keep control over the records with equal PID's the tables with multiple PID's have the records with equal PID's referred to in a Java Vector List.

Vector<DBFRow> pidMultiSet = = new Vector<DBFRow>(6);

TMG tables are therefore stored in a reference structure that makes the data indexed and easily accessible by index and row number (see the next section of this document).

class HREdatabaseHandler

Provides access to the HRE database and the H2 database tables. Typical code:

```
connectString = "jdbc:h2:" + urlH2loc + ";IFEXISTS=TRUE";
dataH2conn =  DriverManager.getConnection(connectString, userId, passWord);
```

The variable dataH2conn is used in SQL SELECT to extract data from the HRE database.

class HREloader

Creates ResultSet objects for each HRE database table. Typical code:

```
TMGglobal.T101 = tableLoader("T101_SCHEMA_DEFNS");
TMGglobal.T104 = tableLoader("T104_TABLE_DEFNS");
TMGglobal.T204 = tableLoader("T204_LABEL_TRANS");
```

class TMGpass_Names

Handles the first pass for population of HRE tables T401 and T402.

class TMGpass_Events

Handles the second pass for population of HRE tables T452_EVENTS.

class TMGglobal

Contains global reference data for the TMG-HRE converter.

Example of pointers to TMGtableData:.

```
      public static TMGtableData  tmg_$_table = null;
      public static TMGtableData  tmg_D_table = null;
      public static TMGtableData  tmg_G_table = null;
```

Example of pointers to HRE table ResultSet:

```
      public static ResultSet T101 = null;
      public static ResultSet T104 = null;
      public static ResultSet T204 = null;
```

class HREexception

Class for exception handling and used for throwing exceptions in the converter.

5. Access to TMG table data loaded into TMGtableData

Class TMGtableData is used to store data for TMG tables and make the data accessible for the conversion process. Each TMG database table is stored in one TMGtableData object. TMG table data are read by DBFReader row by row and each DBFRow object is stored with a reference in the TMGtableData object.

There are two alternative reference lists used to keep control of the DBFRow objects. First the DBFRow object pointers are stored in an ArrayList where the DBFRow can be found and read based on the row number. Secondly the DBFRow object pointers are put into a HashMap with the key PRIMARY ID in the TMG table. Then the DBFRow is indexed and accessible with a HashMap get statement.

```
HashMap<Integer,DBFRow> PIDmap = new HashMap<Integer,DBFRow>();
```

For some TMG tables the PRIMARY ID can be the same for more than one row in the table. For those tables a special solution is made so that the rows with equal PRIMARY ID are referenced in a Vector list and indexed in their own HashMap for the Vectors. This HashMap is created by calling a method in TMGtableData that creates the HashMap for the Vectors for the PRIMARY ID rows with identical value.

```
HashMap<Integer,Vector<DBFRow>> multiPidMap = new HashMap<Integer,Vector<DBFRow>>();
```

Then the vector with equal PID’s in a DBFRow is indexed and accessible with a HashMap get statement.

The monitor printout from loading TMG name tables looks like:

```
 TMG: sample_$.dbf rows: 64 - mrows:  0
 TMG: sample_N.dbf rows: 112 - mrows:  64
 TMG: sample_NPV.dbf rows: 454 - mrows:  112
 TMG: sample_NPT.dbf rows: 11 - mrows:  0
 TMG: sample_ND.dbf rows: 87 - mrows:  0
```

Here “rows” is the total number or row in the TMG table while “mrows” in the number of vectors generated each referring to rows with equal PID. Here in the example sample_N.dbf has 112 rows and 64 vector lists pointing to rows with equal PID’s.

The example below shows how to find the string value in field "SRNAMESORT" in the sample_N.dbf table:

Parameters are:

int listIndex	(row number in the table)
int PID	(row PID in the table)
int vectorIndex	(index in the vector list with equal PID
String FieldName	(the field/column name in the table).

The method to be used for Strings is:

```
    String getValueString(int listIndex, String fieldName);
    String findValueString(int PID, String fieldName);
    String findVectorString(int PID, int vectorIndex, String fieldName);
```

Data in the table is accessed as, for example:

```
    String personName = tmgNTable.getValueString(listIndex, "SRNAMESORT");
    String personName = tmgNTable.findValueString(rowPID, vectorIndex, "SRNAMESORT");
    String personName = tmgNTable.getValueString(rowPID, "SRNAMESORT");
```

6. The HRE database representation

HRE uses the H2 database system, however the HRE software is implemented with an open SQL data interface. The H2 database has all tables in one database file where the individual tables can be presented via the H2 database console on a standard browser.

The library used to access the H2 database allows sending standard SQL requests (SELECT * FROM tableX) to the database and the received data is a ResultSet object that contains the requested data, namely the H2 database table. The ResultSet object also contains the header information for the columns.

The Java library can do standard SQL operation on the ResultSet as CREATE, READ, UPDATE, INSERT and DELETE. It is important to know that the ResultSet copy of table data is not automatically updated from the database when a database SQL operation is done on a table. Only the H2 database is updated.

Doing a ResultSet HREtable insert: HREtable.moveToInsertRow();

Doing an update of new records is, for example:

```
HREtable.updateBoolean("IS_SYSTEM", false);
HREtable.updateBoolean("HAS_CITATIONS", false);
```

An insert with: HREtable.insertRow(); only updates the H2 database table and the ResultSet data remains as it was before the insert.

```
HREtable.refreshRow();
```

refreshes the data in the result set to reflect any recent changes in the database.

The ResultSet class includes get and update methods for retrieving and updating of individual fields in a table row, either by identifying the individual fields by index of the column number from the left or the column header label.

7. Implemented functions in class TMGHREconverter

The following is a description of processing controlled by TMGHREconverter in the TMG to HRE converter.

Class TMGHREconverter is the main controller of the conversion process. TMGHREconverter also has a main method for starting Java that can be used to start the converter independent of the HRE program.

The following tasks are done in TMGHREconverter:

1. Read and present the content of the TMG project info file (.pjc) to check the TMG database version, which must be 10 or 11

2. Load the TMGtables with class TMGloader. This process is described in the section "Access to TMG table data loaded into TMGtableData". For each TMG table to be loaded the following setup is used:

```
// DATASET TABLE D
DBFReader tmgDreader = tableLoader(tmgTableNamePath.dbf);
setMemoFile(tmgDreader, tmgTableNamePath.fpt");
// Store table data in TMGtableData object
TMGglobal.tmg_D_table = new TMGtableData(tmgDreader,"DSID");
```

   If the table (e.g. PPV.dbf) has an equal PID (e.g. Place Part Value), the following extra call is used to generate a Vector:

```
//PLACE PART VALUE TABLE
DBFReader tmgPPVreader = tableLoader(tmgTableNamePath.dbf);
TMGglobal.tmg_PPV_table = new TMGtableData(tmgPPVreader,"RECNO");

// Open for reading of multiple equal indexes
tmgPPVreader = tableLoader(tmgTableNamePath.dbf);
TMGglobal.tmg_PPV_table.TMGtableMultiData(tmgPPVreader,"RECNO");
```

   The variables for pointing to TMG tables in TMGglobal are:

```
public static TMGtableData  tmg_$_table = null;
public static TMGtableData  tmg_D_table = null;
public static TMGtableData  tmg_G_table = null;
public static TMGtableData  tmg_N_table = null;
public static TMGtableData  tmg_P_table = null;
public static TMGtableData  tmg_PD_table = null;
public static TMGtableData  tmg_PPT_table = null;
public static TMGtableData  tmg_PPV_table = null;
public static TMGtableData  tmg_T_table = null;
```

3. Load the HREtables into a ResultSet object for each HRE table with HREloader:
   - Open the H2 database
   - Add columns to the Seed database if needed.
   - Request data from an H2 table with "SELECT * FROM Table"
   - Store the pointer to ResultSet for each HRE table in TMGglobal.

   The variables in TMGglobal are:

```
public static ResultSet T104 = null;
public static ResultSet T204 = null;
public static ResultSet T131 = null;
public static ResultSet T401 = null;
public static ResultSet T404 = null;
```

   Next tasks are to initiate the passes that move data from TMG tables to HRE tables.

4. TMGpass_Name
   In the first pass, transfer data from TMG PERSON (_$.dbf) file to HRE table T401_LIFE_FORMS.
   In the second pass, transfer data from TMG NAME (_N.dbf) to HRE database T402_LIFE_FORM_NAMES
   NOTE: this is a preliminary solution with raw transfer of data between tables.

5. TMGpass_Event
   Implemented so far is a search in EVENT (_G.dbf) and access to other tables to search for place data for birth and death events. Place Part Value File (PPV.dbf), Place Part Type File (PPT.dbf) and Place Dictionary File (PD.dbf) are searched for place name elements for birth and death.

   The pass creates content in the T452_EVENTS and T463_EVENT_NAMES tables

6. Close databases.

8. Example code

```
/**
 * addToT401_LIFE_FORMS(long rowPID, ResultSet HREtable)
 * @param rowPID
 * @param hreTable
 * @throws HCException
*/
private void addToT401_LIFE_FORMS(int rowPID, ResultSet hreTable) throws HCException {
      String tmgDate;
      try {
      // moves cursor to the insert row
            hreTable.moveToInsertRow();
      // Update new row in H2 database
            hreTable.updateLong("PID", proOffset +
                    tmg$table.getValueInt(rowPID,"PER_NO"));
            hreTable.updateLong("CL_COMMIT_RPID", null_RPID);
            hreTable.updateBoolean("IS_SYSTEM", false);
            hreTable.updateBoolean("HAS_CITATIONS", false);
            hreTable.updateBoolean("HAS_LINKS", false);
            hreTable.updateLong("VISIBLE_ID", tmg$table.getValueInt(rowPID,"REF_ID"));
            hreTable.updateLong("SUB_TYPE_RPID", null_RPID);
            hreTable.updateString("SURETY", "3");
            int perNoID = tmg$table.getValueInt(rowPID,"PER_NO");
            if (tmgNtable.existVector(perNoID))
                  hreTable.updateLong("BEST_NAME_RPID", proOffset +
                  tmgNtable.findVectorInt(perNoID,0,"NPER"));
            else {
                  hreTable.updateLong("BEST_NAME_RPID", null_RPID);
            }
            hreTable.updateLong("BEST_IMAGE_RPID", null_RPID);
            tmgDate = tmgNtable.getValueString(rowPID,"PBIRTH");
            if (tmgDate.length() == 0 || tmgDate.startsWith("100000000"))
                 hreTable.updateLong("BEST_START_HDATE_RPID",null_RPID);
            else {
                  hreTable.updateLong("BEST_START_HDATE_RPID",
                  HREhdate.addToT751_HDATES(T751, tmgDate));
            }
            tmgDate = tmgNtable.getValueString(rowPID,"PDEATH");
            if (tmgDate.length() == 0 || tmgDate.startsWith("100000000"))
                  hreTable.updateLong("BEST_END_HDATE_RPID", null_RPID);
            else {
                  hreTable.updateLong("BEST_END_HDATE_RPID",
                        HREhdate.addToT751_HDATES(T751, tmgDate));
            }
            hreTable.updateLong("BEST_BIRTH_GENETICS_RPID", null_RPID);
            hreTable.updateLong("LAST_PARTNER_RPID", proOffset +
                 tmg$table.getValueInt(rowPID,"SPOULAST"));
            hreTable.updateLong("BIRTH_FATHER_RPID", proOffset +
                  tmg$table.getValueInt(rowPID,"FATHER"));
            hreTable.updateLong("BIRTH_MOTHER_RPID", proOffset +
                  tmg$table.getValueInt(rowPID,"MOTHER"));
            hreTable.updateLong("BEST_PARENT1_RPID", null_RPID);
            hreTable.updateLong("BEST_PARENT2_RPID", null_RPID);

      //Insert row in database
            hreTable.insertRow();
      } catch (SQLException sqle) {……………………………}
}
```