# HRE Architecture Layers and API design

**HRE Build 23 level**

**Revision history**

| | | |
|---|---|---|
| 2019-11-09 | Nils Tolleshaug | Original draft referring to TMG functions. |
| 2019-12-05 | Nils Tolleshaug | Updated 3.7 Database Layer and 3.8 Database Handler |
| 2019-12-06 | Don Ferguson | Tidied format, grammar and removed duplicate text |
| 2019-12-07 | Nils Tolleshaug | Updated 3.5 Request Manager and 3.6 Request Listener |
| 2019-12-14 | Nils Tolleshaug | Updated exception handling 3.3 Business Layer and 3.7 Database Layer |
| 2019-12-20 | Nils Tolleshaug | Updated 3.2 Business Layer and 3.3 Business Layer API |
| 2020-01-05 | Don Ferguson | Added Commit comments (2.3); clarified AUX file updates; tidied all text |
| 2020-02-19 | Nils Tolleshaug | Added 2.3.1 Project Menu and 3.3.1 Project Menu design |
| 2020-02-22 | Nils Tolleshaug | Added 2.3.2 Tools Menu and 3.3.2 Tools Menu design |
| 2020-08-12 | Don Ferguson | Minor updates to conform with Build 23 |

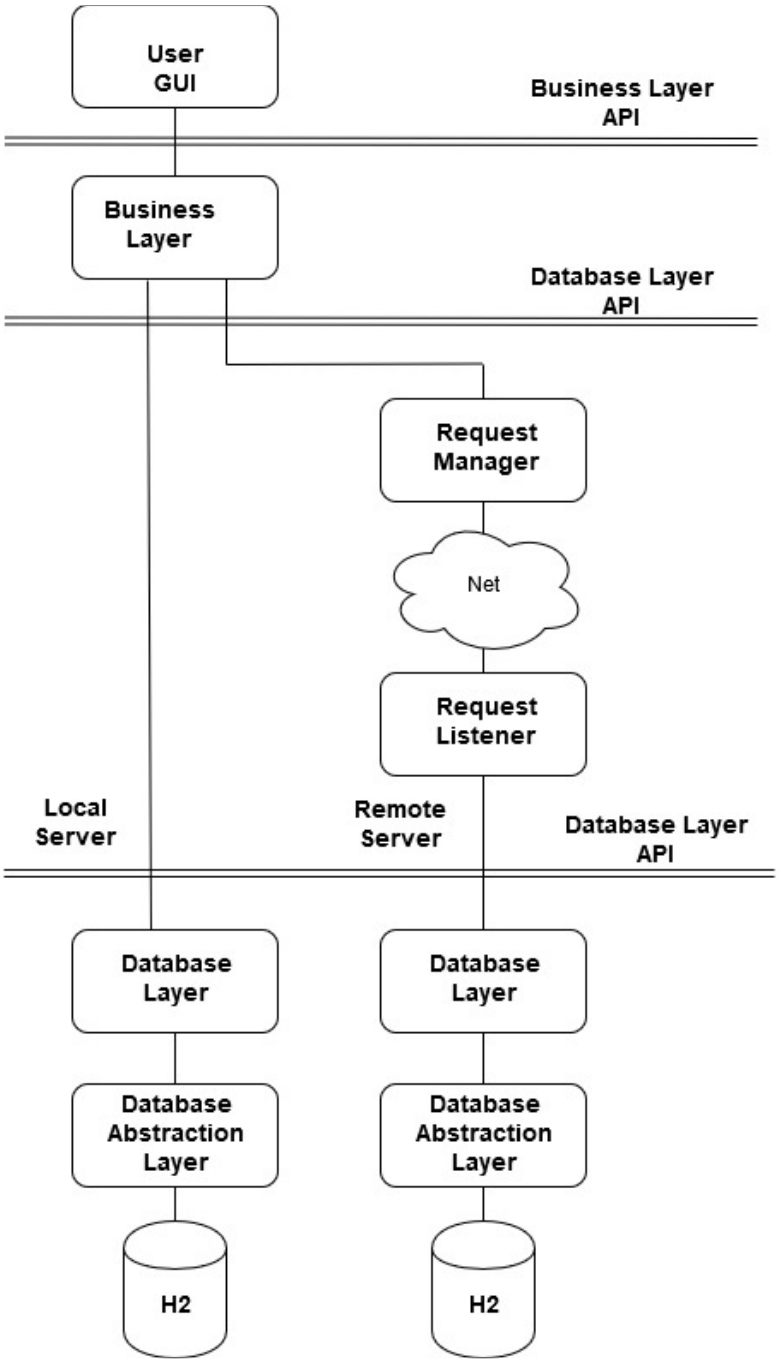
**Table of contents**

# 1 Introduction

This document explains the framework of the HRE software design as at HRE Build 23.

Section 2 includes a functional description of the software layers and the related API interfaces. Section 3 includes design of the layers and the design of the APIs. The Graphical User Interface (GUI) and the GUI elements are programmed in Java classes by using Windows Builder in an Eclipse Java project.

The figure below shows the principal software layers and the APIs described in this document, that are included in the HRE software system.



Data flow starts with an action initiated in the user GUI. A request for data is sent to the Business Layer (BL) where the diagram shows that both in the local and remote server cases, the Database API interface is used to request data from the Database Layer (DL). The design goal is that the Database API interface shall be identical for both local PC and remote server database access. The Database Layer processes the request and collects the requested data set from the HRE database. The Database Abstraction Layer is introduced to support the HRE software system over a standard API hiding the database system (H2) and the database tables of the HRE system.

# 2 HRE Layers and API functions

This section covers the functions that shall be performed in the HRE software system layers and the function of the Application Program Interfaces (API).

## *2.1 Graphical User Interface (GUI)*

This is an overall description of how the GUI user elements are populated with data and how the GUI class objects will be interacting with the Business Layer (BL) class objects. HRE Business Layer delivers data to the GUI on request from the user. Each of the menu elements in the GUI have their own class definition. The HRE specifications include a description and layout of the GUI element.

The BL receives requests for data from the User GUI over the Business Layer API and delivers data to be presented in the User GUI panels. The BL processes the requested data from the HRE project database and prepares window GUI elements, e.g., person tables, family tables and family trees to be added to the user GUI.

## *2.2 Business Layer API*

This is the communication interface between the User GUI menu element and the Business Layer (BL). The BL processes the requests for data, prepares a GUI window element and delivers a link to the GUI data element stored in the BL. Other types of actions initiated from the GUI are commands such as an update of the database. In this case the BL will respond with a confirmation report over the Business Layer API. The BL also handles transfer of sequential file data (such as reports) to be presented as printouts streamed to a printer or separate GUI windows.

## *2.3 Business Layer*

The Business Layer (BL) is the main software layer that delivers data to the user GUI and requests data from the HRE database to be compiled into GUI presentation windows.

The Business Layer (BL) shall administer the operations in the HRE project database and shall manage operations on databases either from an HRE database located on the user client PC or over any TCP/IP network (LAN or WAN) from a remote located server storing an HRE project database.

HRE project administrative tasks shall be administrated by the BL e.g., handling user related data and project related data. This data is not part of the database but is stored in auxiliary files (AUX) in the local client file system or with the remote server database.

The Business Layer allows opening of more than one concurrent database and administrates the open databases and the corresponding components displayed in the User GUI window(s).

More complex operations (such as adding a new person to a family) include sequences of CRUD (Create, Read, Update and Delete) operations which are handled in the Database Layer (DL). Report processing is initiated in the Business Layer, however the data retrieval is handled in the Database Layer. The Business Layer (BL) can initiate single CRUD operations in the Database Layer (DL) but changes have to be bound together to form a single Commit operation. At the end of each Commit the database must be left in a stable, consistent state.

The BL processes the received data from the Database Layer and prepares window GUI elements e.g., person tables, family tables and family trees to be added to the user GUI. The Person Selector is a typical window where data presented in the window is prepared by the BL. Also source, name, and place lists are handled by the BL and presented to the User GUI.

To set up a display of family trees, BL initiates a process in the Database Layer (DL) where a DL initiated process with a recursive database call retrieves the data to be used for the Java object family tree structure. The BL processes the received data for presentation in the User GUI.

The BL initiates production and transfer of report files from the HRE database to separate GUI windows or the client print server.

For every open project, the Business Layer shall include a memory copy of the relatively stable database definitions data maintained as Java classes. The aim here is to reduce the number of Joins in the actual user data SQL commands used in retrieval requests.

**Note:** there will be a restriction that the Business Layer will check the database version against the software version and if the versions are incompatible an update of the HRE database will be necessary.

HRE User GUI setup data is also stored in the BL and saved in the local client AUX data. The Business Layer keeps project data and data for user GUI presentation.

Error handling or exception handling is handled by the BL and error messages are presented from the Business Layer.

When Projects are shared by connections to other computers (from the same User) or where the shared access is among two or more Users, there is a need for interactive project status data. For each User the BL must perform updates to the relevant User AUX files. The Project AUX files need to be stored with the HRE Project database on the server, so that another User can retrieve the current status, and so that there can be message sessions between Users of the same project or server. Refer to *03.32 Overview - Auxiliary (Non-Database) Files*.

### *2.3.1 Project Menu*

The following menu items are now implemented. (Compare, Merge and Split are still inactive). A project name is unique and the system does not allow identical project names in the known projects list. A general rule is that if a new project is created, the project is opened.

The project menu has a “Summary” button that shows more info related to the selected project.

- Open Project
  User selects a project from the known project list in the table, selects “Open” and the selected project is opened. The Open project name and database version are shown in a dialogue.
  The user may also click on project names listed under the Project menu’s ‘Recent Project’ list to open the project dirsctly.
  Alternatively, the user can select “Browse” and use the File Chooser to select a database file in a selected folder. Selecting “Open” creates a new project entry in the list of known projects with reference to the selected project database file and project name copied from table T204_LABEL_TRANS in the database

- Create New Project
  User enters the new project name and selects “Browse”. The user selects a folder and a name for the new database project file via the File Chooser menu. A new project is created in the list of known projects, the HRE Seed database is copied to the selected folder and given the selected database filename. The new project name is updated in T204_LABEL_TRANS in the database and the table T131_USERS is updated with the user data from HGlobal.userCred (the user’s credentials)

- Backup Project
  User selects a project from the known project list. In the section BACKUP TO the user selects "Browse" and selects a folder to store the backup file. Selecting "Backup" causes the database to be zipped and copied into a backup file with the project name as filename and date/time for the backup in the last part of the file name with extension .hrez. If the project is open, the user is warned that the project will be closed before the backup

- Restore Project
  User selects a project from the know project list in the table. In the 'From Backup' section the user selects "Browse" and selects with file chooser the backup file to restore from. If user selects "Restore" the user will be warned that the selected project database file will be overwritten and the process cannot be undone. Answering YES cancels the restore operation.
  Alternatively the user can use "Restore to a new project" and select "Browse". With file chooser the user selects a folder for the restored database file. The user enters the name of the new project and selects "Restore". A new project is created with the new project name, the database file is restored to the selected folder and the filename is copied from the filename of the selected backup file

- Close Project
  User selects a project from the open project list. "Close" closes the connection to the HRE database. Alternatively, clicking "X" on the main screen will invoke this screen, except when only one project is open, in which case it will be closed immediately

- Copy Project As
  User selects a local or remote server and selects "Browse" to select a database file name with the file chooser. In the "Copy To" section the user enters the name of a new project and selects "Browse" to select the folder for the destination file. By selecting "Copy", the chosen database file is copied to the selected folder and a new project is created and included into the known project list. The project name in the database file table is not updated. "Rename Project" must be used to update project names

- Rename Project
  User selects a project from the available project list, then enters the new project name and selects "Rename Project". The project name in the known project list in the User AUX file and the project name in table T204_LABEL_TRANS in the database are updated with the new project name

- Delete Project.
  User selects a project from the available project list. The project must be closed before delete. User selects "Delete" and is asked if the database file shall be kept. If the answer is YES the database file is not deleted

- Exit HRE
  When user selects "Exit HRE" the user is asked to close all open projects.

### *2.3.2 Tool Menu*

The following menu items are implemented

- Tools/Setting/User
  The user can set user options including change of menu language between English, French and Norwegian and various other user and GUI-related options. "Accept" activates the changes

- Tools/Administration/Edit User Rights
  Shows user data for a project from T131_USERS in the database. Add, Change and Delete of database User records are implemented. Setting of User Rights is not implemented.

## 2.4 Database Layer API

In the local case this API handles data transport between the BL and Database Layer (DL). In the remote case this API also will also be used to communicate with a Request Manager in the client which is connected over the network to a Request Listener on the server.

The API interface shall be as much as possible identical for operation on local database on the client PC and remote databases on a remote server. This implies that the API will be used for communication to remote servers and should not include API functionality that cannot easily be implemented in client/server communication.

The Database API specification includes a list of actions to be submitted over the interface and a description of the corresponding data received as response to the action. There will be three main types of requests to the Database Layer:

- initiation of presentations for the GUI that need a sequence of database operations
- initiated report production and transfer to the User GUI
- transfer of pictures or maps from the database.

The API needs to transfer exception handling from the Database Layer to the Business Layer.

## 2.5 Request Manager

Request Manager receives from the Business Layer a request for data from a HRE project located on a remote server. The function of the Request Manager and Request Listener is to transparently handle request and response data over the Database API, as in the Local server case. The Database API call contains all parameters needed to extract the relevant data from processes in the Database Layer.

RM shall need to report Exceptions to the Business Layer in case of interrupted communication with Request Listener RL.

The RM may also need a life pulse communication with the Request Listener for transparent communications.

## 2.6 Request Listener

Request Listener (RL) handles the request from the Request Manager and converts the request into a Database Layer API call. The received data from the Database Layer are converted to a serial stream and sent over the network to the Request Manager.

## 2.7 Database Layer

Functions located in the Database Layer (DL) include tasks that need to be done on the server. Typical functions are requests that involve extensive use of a series of SQL calls to the database.

Advanced database operations e.g., processing of reports, is initiated by the DL and handled by processes accessing and addressing the HRE database with CRUD (Create, Read, Update and Delete) commands to the Database Abstraction Layer. CRUD includes the following operations:

- Create a standard CRUD command SQL request
- Send data request to Database Abstraction Layer
- Receive data or confirmation from Database Abstraction Layer.

Administrative tasks on HRE project databases such as copy, split and merge of HRE project databases are considered as complex database operations and should be handled by additional class definitions extending the class Database Layer.

Examples of DL operations are: report generation, family tree presentation, copy/split/merge of databases, database maintenance (consistency checks), and searches for person duplicates, etc.

Single CRUD (Create, Read, Update and Delete) record operations can be done transparently from the client with standard SQL calls to the Database Abstraction Layer, provided Commit functions are also single record operations.

File transfer of documents and pictures referred to by the database are other non-database modifying actions that need processing in the server.

## *2.8 Database Abstraction (DA)*

The function of Database Abstraction is to separate the HRE code from the database engine and command language. The DA should be able to hide the database engine from other parts of HRE software, so that replacement of the database engine or minor changes in HRE database tables, do not demand an update to the HRE software layers.

HRE have selected H2 as the SQL database engine, however it is foreseen that the HRE project should be able to change to another SQL database engine if H2, for some reason or other, becomes unsupported in the future.

The DA shall also be used in case changes in the HRE database table design require an update of the SQL database commands.

However there will be a restriction that the Business Layer will check the database version against the software version and if the versions are incompatible, an update of the HRE database will be necessary.

# 3 HRE Layers and API Design

This section gives an overall description of the class structure, methods and software blocks in the HRE software system.

## *3.1 Graphical User Interface (GUI)*

HRE Business Layer delivers data to the GUI on request from the User. Each of the menu elements in the GUI have their own class definition according to HRE specifications, but it is not specified how the GUI user elements are populated with data and how the GUI class objects will be interacting with the Business Layer class objects.

The list of GUI Specifications can be found on GitHub.

## *3.2 Business Layer API*

The User's GUI sends a request for data and receives (for example) a pointer to a Java Swing JPanel to be included in the User GUI JPanel. The API will also return other formats of data.

The API has two basic operations - to initiate an operation in the Business Layer and to receive requested data. The returned data will be an instance of a JComponent that can be added to a JDialog, JFrame or JInternalFrame, added to a GUI window and made visible. The subclasses of the JComponent can be a JTable presenting data in a table, a JTree with a family presentation and so on.

The connection between User GUI and the Business Layer is created with pointers from the main menu display elements. Each of the 10 main menu User GUI items has a corresponding handler in the Business Layer. Therefore the Business Layer has 10 main handlers each processing action requests from the main User GUI windows.

Each Handler is a sub class of class BusinessLayer such as:

```
class HBProjectHandler extends HBBusinessLayer
```

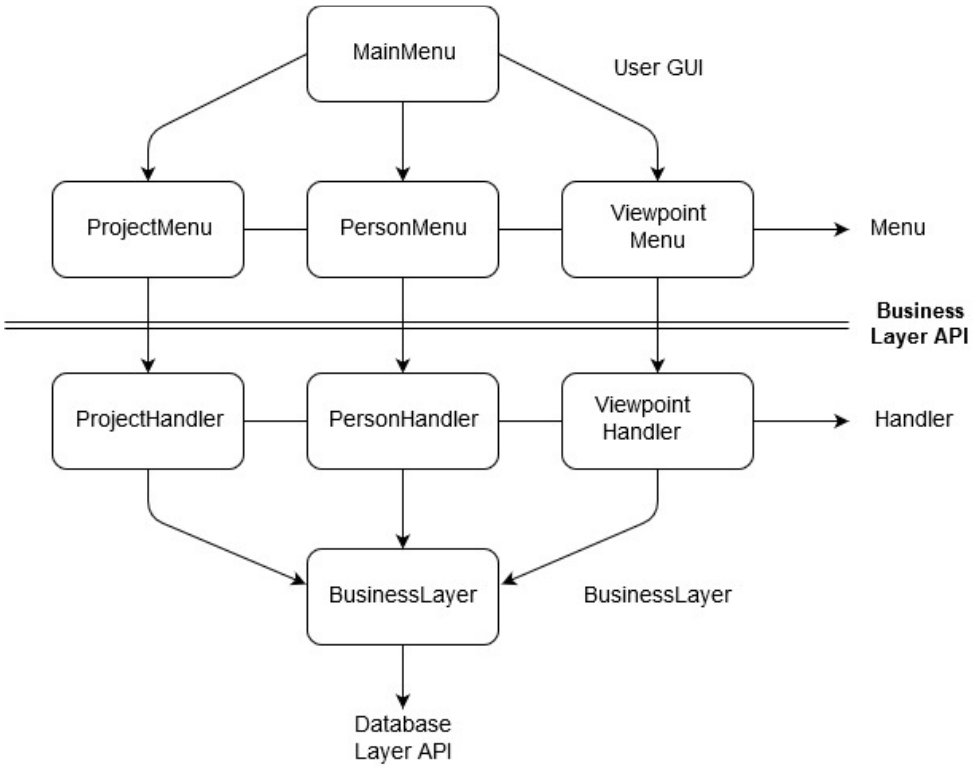
The pointers to BusinessLayer handlers are created during startup as:

```
pointBusinessLayer[0] = new HBProjectHandler();
pointBusinessLayer[1] = new HBPersonHandler();
pointBusinessLayer[2] = new HBResearchHandler();
pointBusinessLayer[3] = new HBViewpointHandler();
pointBusinessLayer[4] = new HBEventTaskHandler();
pointBusinessLayer[5] = new HBWhereWhenHandler();
pointBusinessLayer[6] = new HBEvidenceHandler();
pointBusinessLayer[7] = new HBReportHandler();
pointBusinessLayer[8] = new HBToolHandler();
pointBusinessLayer[9] = new HBHelpHandler();
```

The pointers are initiated in the User GUI screens as:

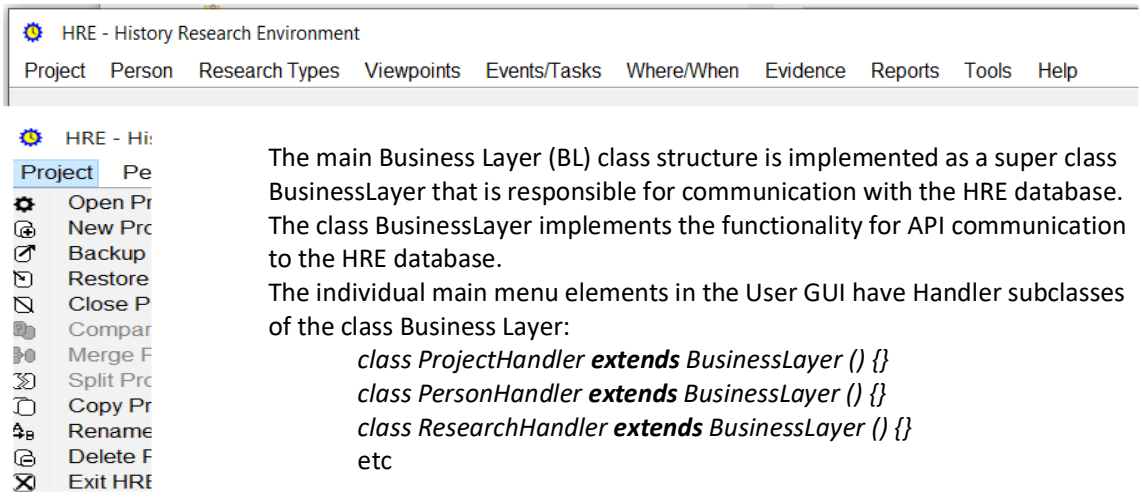
```
new HG0402ProjectOpen((HBProjectHandler)pointBusinessLayer[0]);
```

The following diagram shows the building blocks and pointers connecting User GUI and Business Layer over Business Layer APIs.




## *3.3 Business Layer*

Each main menu item such as Project, Person, Research Types etc. shall have their own class definition.




The main Business Layer (BL) class structure is implemented as a super class BusinessLayer that is responsible for communication with the HRE database. The class BusinessLayer implements the functionality for API communication to the HRE database.
The individual main menu elements in the User GUI have Handler subclasses of the class Business Layer:

*class ProjectHandler **extends** BusinessLayer () {}*
*class PersonHandler **extends** BusinessLayer () {}*
*class ResearchHandler **extends** BusinessLayer () {}*
etc

The BusinessLayer can open more than one database and data for open projects are stored in an object instance of class OpenProjectData that has a database index of the corresponding open database controlled by DatabaseLayer. The instances of class OpenProjectData is referred to from an ArrayList called openProjects, located in the class HGlobal.

In the example above, the class ProjectHandler will inherit all methods and data in class BusinessLayer.

The handler executes all action methods initiated from the User GUI window elements like this:

```
if (pointProHand.openProjectAction(selectedProjectName)) dispose();
```

Then the ProjectHandler method will have the following code:

```
public boolean openProjectAction() {
        // process open project and database
        if (successful_open) return true;
        return false;
}
```

If the "action" method returns "true" the User GUI window is closed – or disposed()

Class BusinessLayer has, in the local database case, a pointer to the Database Layer and calls methods in the Database Layer.

The Business Layer performs exception handling and "catch" exceptions from the Business Layer (BL) as instances of class HBException and "catches" exceptions from the Database Layer (DL) as instances of class HDException. Each method in BL and DL "throws" the exception upwards to BL where the exception is reported to the user. The following code example explains the exception handling principle in the Business Layer:

```
public HBProjectHandler() {
      super();
      try {
            pointDBlayer.connectSQLdatabase("H2");
      } catch(HBException exc) {
            System.out.println("Project Handler - init: \n" + exc.getMessage());
                  JOptionPane.showMessageDialog(null, "Business Layer error!\n"
                              +  exc.getMessage());

      } catch(HDException exc) {
            System.out.println("Project Handler - init: \n" + exc.getMessage());
                  JOptionPane.showMessageDialog(null, "Database error!\n"
                              +  exc.getMessage());
      }
```

See Section 3.7 Database Layer for details of how the HDException is generated in the DL.

### *3.3.1 Project Menu*

Each project menu item is associated with a projectAction method in class HBProjectHandler extends HBBusinessLayer, e.g. openProjectAction is the action method for menu Project Open.

Class HBProjectHandler inherits methods and data from HBBusinessLayer where the constructor creates and sets up pointers to class HBLibraryResultSet and class HBLibraryBusiness as follows:

```
      public HBBusinessLayer() {
/**
 * Set up pointers to library class objects
 */
            pointLibraryResultSet = new HBLibraryResultSet();
            pointLibraryBusiness = new HBLibraryBusiness(this);
      }
```

Therefore in the current setup each menu handler will have a separate instance of both class HBLibraryResultSet and class HBLibraryBusiness.

Copy file with Progress Monitor is implemented for the "Copy File As" menu. The copy file process - HBCopyFileProcess is started in a separate Thread and sends event messages to HG058705ActionProgress which is also running in a Thread. The event message triggers progress bar updates in HG058705ActionProgress. The following code initiates the copy process.

Note: Copy file process needs to be the last action point in HBProjectHandler action, since the copy process cannot synchronize with the action process.

```
/**
 * Start CopyFile as new Thread with update to Progress Monitor
 * @param sourceFile
 * @param destinationFile
 * @throws HBException
 */

      public static HG058705ActionProgress pointProgressDisplay;
      public void setupUpCopyFile (File sourceFile, File destinationFile) throws
      HBException {
            pointProgressDisplay = new HG058705ActionProgress();
            pointProgressDisplay.setModalityType(ModalityType.APPLICATION_MODAL);

      // Position set from parent window and stored in HGlobal.ButtonXY
            Point xy = HGlobal.ButtonXY;   // Get button location
            pointProgressDisplay.setLocation(xy.x+100, xy.y-200);

      // Start new Thread with Progress Monitor
            Thread monitor = new Thread(pointProgressDisplay);
            monitor.start();

      // Start copy process as new Thread
            HBCopyFileProcess task = new HBCopyFileProcess(sourceFile,
                        destinationFile,this);
        task.addPropertyChangeListener(pointProgressDisplay);
        task.execute();

      }
```

### *3.3.2 Tool Menu*

The following menu items are implemented in class HBToolHandler extends HBBusinessLayer which has pointers to class HBLibraryResultSet and class HBLibraryBusiness that have methods called by action methods in HBToolHandler.

- Tools/Setting/User
  Change menu language between English, French and Norwegian (calls language setup in Main menu)
- Tools/Administration/Edit User Rights
  Show user data from T131 in database. Add, Change and Delete implemented. Database is modified with the use of interface ResultSet that contains a copy of the database table as a Java object. A modification in the ResultSet is copied to the database.

The following code is an example of how ResultSet is used to insert a new row in a database table. The variable sqlTable holds a copy of table T131_USERS in the connected HRE database:

```
// Increment PID for next row i table
      sqlTable.last();
      long valuePID = sqlTable.getLong("PID");
      valuePID++;
      sqlTable.moveToInsertRow(); // moves cursor to the insert row

// Update new row in ResultSet
      sqlTable.updateLong("PID", valuePID);
      sqlTable.updateLong("CL_COMMIT_RPID", 0);
      sqlTable.updateString("LOGON_NAME", userData[0]);
      sqlTable.updateString("USER_NAME", userData[1]);
      sqlTable.updateString("PASSWORD", userData[2]);
      sqlTable.updateString("EMAIL", userData[3]);
      sqlTable.updateLong("USER_GROUP_RPID",1000000000000001L);
      sqlTable.insertRow();
      sqlTable.beforeFirst();

// Update of table T104 is not implemented
      sqlTable.close();
```

## *3.4 Database Layer API*

The Database API receives requests from the Business Layer. The request for data is converted to a Database Layer API function and receives data converted to Document Object Model (DOM) structure or to a one- or two-dimensional object Array.

List of example operations in the Database Layer:

- Family tree presentation
- Tables of persons, events etc
- Retrieve Image from database
- Sequential file transfer with report data
- Copy, split and merge databases
- Maintenance operations (table integrity check, search for duplicates etc).

Database Layer calls initiate CRUD operations in the Database Abstraction Layer (DAL) with parameters (i.e. type of database request and parameters). The method returns a one- or two-dimensional array with the selected data from the database:

- Object [][] databaseRequest(requestType, parameters…) {}
  (Object type array in use because the data can contain different data types e.g.. String, integer, Boolean, float etc.)
- String databaseRequest(requestType, parameters…) {}
- File Object [][] databaseRequest(requestType, parameters…) {}
- Image Object [][] databaseRequest(requestType, parameters…) {}

The Business Layer also “catches” HRE Exceptions and presents the error message to the user.

## *3.5 Request Manager*

Request Manager (RM) receives requests for data from the Business Layer over the Database API and converts the request into an HTTPS protocol network request to Request Listener (RL). There will be a separate set of RM instantiated classes for each remote project opened on a remote server.

The request for data is converted to an HTTPS GET or POST call over the network. RM needs to convert each operation handled over the Database Layer API. The requests according to the DL API specification are converted to a HTTPS GET or POST with corresponding parameter values to perform the requested operation. The RM opens a serial read data stream from the incoming IP post to

process the incoming XML formatted data. The incoming XML formatted data stream is either converted to Document Object Model (DOM) structure or to a one- or two-dimensional object Array.

Request Manager needs to have a timeout process to terminate a request if the Request receiver does not respond.

Request Manager and Request Listener should also have an error message transfer function that "throws" an HRE Exception in the Request Manager. The HRE Exception is "thrown" from Request Manager and handled in a "catch" statement in the Business Layer.

## *3.6 Request Listener*

Request Listener (RL) receives the HTTPS request for data from the Request Manager. RL converts the request for data to an API call to the Database Layer according to the Database Layer API specification.

RL listens to the incoming HTTPS port and process the HTTPS request for data into a database request submitted to the Database Layer (DL). The returned data format can be in the form of a ResultSet or an object array. The received data is converted to a serial XML formatted data stream. The serial XML data stream is sent over the network to the Request Manager.

## *3.7 Database Layer (DL)*

The Database Layer (DL) initiates CRUD (Create, Read, Update and Delete) operations in the HRE database. The Database Layer also includes handling of complex processes for handling of database data retrieval including various sequences of CRUD operations as:

- Family tree presentation with recursive database calls
- Tables of persons, events etc
- Retrieve Image from database
- Processing of reports with sequential file transfer.

Other advanced database operations that can be handled in DL are:

- export and import of external databases
- split and merge databases
- maintenance operations (table integrity check, search for a duplicate etc).

It is up to discussion if these complex database operations should be handled in separate class DatabaseLayer maintenance class definitions.

The class DatabaseLayer communicates with DatabaseSQLhandler and requests data from DatabaseSQLhandler.

The following code example from the H2 database handler show how the "catch" of specific exceptions from the method database connect are handled in a try/catch statement and a new HDException with the error message is thrown upwards and each method in the calling "chain" throws the HDException until the exception finally is handled in the Business Layer.

```
public DBLDatabaseH2handler(String urlH2loc) throws HDException {
    super();
    try {
      Class.forName("org.h2.Driver");
      dataH2conn =  DriverManager.getConnection(urlH2loc, userId, passWord);
    } catch(SQLException exc) {
            throw new HDException("Connect SQL error:\n" + exc.getMessage());
    } catch(ClassNotFoundException clx) {
            throw new HDException("Connect SQL error:\n" + clx.getMessage());
    }
}
```

## *3.8 Database SQL Handler (DH) and Database Abstraction*

The Database SQL Handler (DH) receives HRE standard SQL requests for data from the Database Layer where the DL calls methods in the interface DatabaseSQLhandler.

Handling of database implementations are implemented in class DatabaseSQLhandler. These classes implementing interface DatabaseSQLhandler handle specific implementations of an HRE database. For example, HRE databases may be implemented in an H2 database engine or alternatively in a MySQL database engine.

Use of interface DatabaseSQLhandler class:

> ***Class** DatabaseH2handler **implements** DatabaseSQLhandler and handles the HRE H2 database. **Class** DatabaseH2handler has methods for connect, request data and close database connection in H2 database. DatabaseSQLhandler is implements as an **interface** and Database Layer calls methods in DatabaseSQLhandler that are implemented in **class** DatabaseH2handler.*
>
> *Alternative database handlers can be implemented e.g. as class DatabaseMySQLhandler **implements** DatabaseSQLhandler. Class DatabaseMySQLhandler must implement all methods from interface DatabaseSQLhandler and these methods need to follow the rules of the MySQL implementation.*

To hide the HRE software system from small changes or updates of the HRE database table design, database abstraction is implemented as a class DatabaseAbstraction with static methods that modify the SQL request string and the returned data in the ResultSet.

Example:

```
// Adjust SQL requests according to setting in DatabaseAbstraction
String updatedString = DatabaseAbstraction.updateSqlString(sqlSearchString);

return DatabaseAbstraction.updateResultSet(pointResultSet);
```