Alive

1.0

Generated by Doxygen 1.8.3.1

Mon Apr 29 2013 08:52:02

Contents

Chapter 1

Alive Documentation

Author

Boris Bulanek National Radiation Protection Institute, Bartoskova 28, 140 00, Praha 4 boris.-bulanek@suro.cz 00420 226 518 279

Date

02/19/13

1.1 About the program

Program Alive (**A**nalyst **li**ght **ve**rsion) is created in order to provide simple and fast OSL dose estimation. Program allow user to calculate a dose using at least one calibration and one natural signal. Signal is corrected on fading. Definition of a signal value is tested by Plateau test. A binary file from the Sequence reader can be used as an input. Program allow user to obtain and store data using Sqlite database.

The program uses some additional packages:

- UI (User Interface) framework Qt
- a package for minimization GSL
- collection of C++ tools Boost
- database software called Sqlite

1.2 Installation

The program has been created on Linux. A single executable file **alive.exe** is created for Windows (7 or XP) users using static linking. Linux users can execute `alive.exe` using program wine. I haven't tried to install the program on Mac but I believe that the installation is similar to installation on Linux.

1.2.1 Linux

All of needed packages have to be in official repositories of your distribution as all of them are open source programs. In case of Arch Linux distribution, you have only to write down command like:

```
sudo pacman -S cmake boost boost-build boost-libs gsl qt sqlite3 sqliteman.
```

After installation you have to go to the src directory and e.g. follow these steps:

```
mkdir build
```

```
cd build
qmake-qt4 ..
make
./alive
```

This approach uses shared library linking where the size of the executable `alive` is considerably smaller and you can use all of additional packages mentioned in the beginning of the section.

1.2.2 Sqlite database

All the information from binary file and some additional parameters can be stored in a simple database called `Sqlite`. There exist graphical tool for working with a Sqlite database called `Sqliteman`.

1.3 Program user guide

1.3.1 Overview

This program has an ability to estimate the dose from OSL measurement using TL/OSL `Risø software` binary output provided by Sequence reader. In comparison with Analyst program, it provide possibility to select calibration runs, fit them with proper calibration curve, and estimate dose of selected measurement without need of regeneration dose signal data. Fading correction using different fading functions are included as well. As an output is used Sqlite database. User can dump all data to txt files as well.

1.3.2 Algorithm

The signal is defined as a sum of first N channels signal value minus signal of the next N channels due to background substraction. Default falue is $N = 20$. Error of signal in channels is given by Poisson distribution. In order to compute dose of the "Natural" signal, there is minimum one "Dose" signal with known irradiation time required. User can choose functions following calibration functions:

- linear $a + bx$; used by default; in case of one calibration point only $a = 0$
- quadratic $a + bx + cx^2$
- cubic $a + bx + cx^2 + dx^3$
- exponential $a * \exp((x - b)/|c|)$
- line+exponential $a * \exp((x - b)/|c|) + d * x$

Parameters $a, \ldots, d$ are written in program as $[0], \ldots, [3]$.

The proper parameters of function are estimated using χ^2 minimization. The error of the dose is computed using assumption of linear behaviour of function at the point of the signal in error range: $\delta(dose) = \delta(signal)/\frac{\partial f(x)}{\partial x}|_{f(x)=signal}$.

The fading function value is defined as a value which you have to multiply by the hypothetical unfaded signal value in order to get the measured (faded) value.

User can choose one of three possible fading functions for every measurement:

- constant a used default with $a = 1$
- logarithmic $a - b\ln(t)$

- exponential $a * t^{-b}$, where t is the time between irradiation and measurement.

  Fading parameters a, b are written in the program as fad_0, fad_1. User can adjust as the fadin parameters as the time-unit. Dose D is corrected by fading of signal itself and fading of the calibration signal using formula:

  $D_{corrected} = \frac{F_{calibration}}{F_{natural}} \cdot D_{uncorrected}$, where F is previously mentioned fading correction. Because of assumption of the same time between calibration irradiation and measurement of the signal for all the calibration points there is no separate correction of calibration signal. It can be easily changed in the future.

1.3.3 Running program

After executing alive executable (on Windows in xterm terminal) and open file (database) through `File->Open File (Db)` you will see

1. Graphs of all measurements in the binary file (database).

2. The table of single measurements information. You can select, which variable to show by clicking on the button `Variables`. Then you will see the window with all the variables. They are divided in two groups. First group are the variables, which are stored in the origin binary file created by `Sequence Viewer`. The second group are variables created for this program. While clicking and staying with the mouse on the name, you can see the help information. This information is provided by the manual for `Analyst` by Risø. The same names of variables are used while saving information into `Sqlite` database.

You can select a subset of measurements for analysis editing a line `Edit Measurement` in a main window. You can select as a single measurement as a range of measurements (e.g. 1,2-4). In order to adjust informations as irradiation and measurement time due to fading correction or irradiation time and type of data for calibration and subsequent dose estimation, you have to edit the needed variables by selecting proper measurement number and clicking on `Edit Measurement` button. After adjusting of all parameters, you can see the signal-dose graph with calibration fit by clicking on `Dose estimation` button. You can see the table with variables needed to estimate uncorrected signal from data(signal and background range), fading correction (fading function and parameters) and informations about data type and irradiation time. You can select time unit of time in a fading function, show uncorrected signal data by clicking on `Show signal data`, change calibration function, show parameters of calibration function (you have to click on replot in order to change the parameters) by clicking on `Fit result` button. By clicking on `Compute Dose` button and writting down the particular measurement, you will see the table of computed dose with(out) fading correction and fading correction itself.

In order to save data for further processing you can save all the data to the txt file by clicking on `File->Save txt` in a main window. Until now there is not a possibility to read such a txt file for further processing. But you have a possibility to store and reuse of all data by using the `Sqlite` database. In order to save all the information to a database, you have first to create the database. You can save the database by clicking `File->Save db` in main window. You will see the window where you have to type the name of the database and in case of saving data the name of database entry. After typing the database name and clicking on the `Connect to database` button you will see the table of all already saved binary files. Right clicking on the particular line, you will see the possibility of deletion of `Sqlite` data from particular binary file with deletion of all subtables or looking deeper on measurements for particular saved binary file. After right clicking on the particular measurement, you can look on the measurement data. All the tables are editable and by editing them you are changing directly the information in the `Sqlite` database. More about organization of database tables is written in section Database tables.

1.3.4 Plateau test

Clicking on the "`Show Plateau`" button in the main window, you can see the graph of the dependence of the calculated uncorrected dose on the range of a signal and background interval. The range value `VALUE` on the x-axis means that the signal for the dose calculation is evaluated as integral (0,`VALUE`) minus background integral (`VALUE+1`,2*`VALUE+1`).

1.3.5 Database tables

All of the tables are generated automaticaly while saving the database. Tables in the `Sqlite` database are divided into three levels. First-level table called INFO_1 is a table of saved binary files. Every line is connected to informations of particular binary file. This table included only information like ID(identification number, automaticaly incrementaly generated from 1), number of measurements, time of saving data. Second-level of tables called INFO_2_<ID> are tables of information for particular file where ID is mentioned identification number. Every line is connected to information of one single measurement of original binary file. This tables include information like data type of measurement, luminiscence type, irradiation time etc. The third level of tables called INFO_3_<ID>-_<num_measurement> are tables of data of particular measurement where num_measurement is a number of measurement of particular database entry from binary file with identification number ID.

E.g. if i've saved two binary files into `Sqlite` database and want to see data of the fourth measurement of the second binary file, have a look into `INFO_1` table and look on the ID of the second measurement (have to be 2), and show the `INFO_3_2_4`. If you are using the `Sqlite` database window table of the program (`File->Open db` or `File->Save db`) you can show this table by clicking on the line with the correct ID in INFO_1 table and then on the line of the right measurement in the second-level table.

There is a possibility to dump all the information from run to the txt file.

1.4 Developement of the program

The code is developed using C++ programming language and files developed in `C++` are all file with .cc (source) and .h (header) sufix. Files with .ui sufix are created by using Qt designer tools for UI (User interface) developement. Compilation generate files with ui and moc prefixes (more `here`).

The documentation of the code is created by `doxygen`. This document page is created by it as well. Clicking on the `"Data Structures -> Class Hierarchy"` at the top of this page you can see the objects (classes) of this program. Following is the brief description of all classes.

1.4.1 Program structure overview

Obtain data from binary file is provided by BinReader class. Storage of data for one single measurement is provided by Data class. Computing of needed observables asi the dose and fading correction is provided by DataHandle class. Reading and writting the `Sqlite` database is provided by SqlHandle class.

Classes which provide the user interface are classes inherited by QMainWindow and QDialog. There are also .ui files with the same name providing the design of windows only.

- MainWindow is the class of the window, which you can see after executing the program executable.
- Info is connected to the window, which you can see after clicking on Edit Measurement button in MainWindow.
- Calibration is connected to the window, which you can see after clicking on Dose estimation button in MainWindow
- Plateau is connected to the window, which you can see after clicking on Plateau button in MainWindow
- SqlConnection is connected to the window, which you can see while opening or saving `Sqlite` database
- DbSave is connected to the window with progress bar, which you can see while saving the database entry.
- MyThread is a special class without .ui file devoted to changing the progress bar in DbSave class while saving the database entry.

There is also the directory `dlib` where is stored the C++ code for fitting. It will be removed soon using `GSL` library which also provides covariance matrix of parameters, so the error estimation of fitting parameters as well.

In order to develope program, you have to develope an user interface as well. For that purpose you can use `QtDesigner` by typing in terminal:

```
./designer(.exe)
```

The developemnt tool for both code and user interface design developement which I've used is `QtCreator`. Unfortunately I couldn't find the proper way to install it on Windows until now. It can be installed without problems following the installation guide from the installation executable from the `web page` for Windows, but there is a problem with a program integration (compilator integration, Qt integration).

The code itself has not been documented until now. Everybody is welcome to help with the code developement. The developement of the code and documentation can and be managed using `git` version system and all the code with possible developement branches stored in `https://github.com/bulanek/alive`. In order to commit your code and documentation changes to the web page which will be the storage of the newest code with all the history changes and possible branches you have to create (free) `GitHub` account and send me an email with your user name. I will then put yourself to the list of contributors.

If you have any question, please send me a mail to: `boris.bulanek@suro.cz`

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

Namespace Documentation

5.1 Ui Namespace Reference

5.1.1 Detailed Description

Author

Boris Bulanek National Radiation Protection Institute, Bartoskova 28, 140 00, Praha 4 boris.-bulanek@suro.cz 00420 226 518 279

Date

01/30/13

Author

Boris Bulanek () National Radiation Protection Institute, Bartoskova 28, 140 00, Praha 4 boris.-bulanek@suro.cz . 226 518 279

Date

01/30/13

Chapter 6

Data Structure Documentation

6.1 BinReader Class Reference

Public Member Functions

- **BinReader** (const char ∗fileName)
- vector< Data > **setData** (const char ∗fileName)
- struct tm & **getDateTime** (const unsigned which)
- string **getPascalString** (FILE ∗inFile, const int aSize) const
- const vector< Data > & **getData** () const

Friends

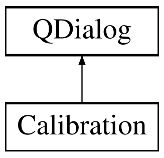
- ostream & **operator**<< (ostream &stream, const BinReader &baseClass)
- ostream & **operator**<< (ostream &stream, const BinReader ∗baseClass)

The documentation for this class was generated from the following file:

- /home/boris/Dropbox/dokumenty/SURO/EURADOS_school/intercomp/my_program/qt_application/alive/src/Bin-Reader.hh

6.2 Calibration Class Reference

Inheritance diagram for Calibration:



Public Member Functions

- **Calibration** (QWidget ∗parent=0)
- Ui::Calibration ∗ **getUi** ()
- void **showData** ()
- void **savePlot** ()

- void **getCalibrationParameters** ()
- void **showTable** ()
- void **saveTableData** ()
- virtual void **accept** ()

The documentation for this class was generated from the following file:

- /home/boris/Dropbox/dokumenty/SURO/EURADOS_school/intercomp/my_program/qt_application/alive/src/calibration.-h

6.3 Data Class Reference

Public Member Functions

- Data & **operator=** (const Data &data)
- string **getValue** (const string aVar) const

Static Public Member Functions

- static const map< string,
  string > & **getVarNamesHelp** ()

Data Fields

- short **Version**
- short **Length**
- short **Previous**
- short **NPoints**
- int **LType**
- float **Low**
- float **High**
- float **Rate**
- short **Temperature**
- short **XCoord**
- short **YCoord**
- short **Delay**
- short **On**
- short **Off**
- int **Position**
- int **Run**
- char **Time** [7]
- char **Date** [7]
- string **Sequence**
- string **User**
- int **Dtype**
- float **IRR_Time**
- int **IRR_Type**
- int **IRR_Unit**
- float **BI_Time**
- int **BI_Unit**
- float **An_Temp**
- float **An_Time**

- int **Norm1**
- int **Norm2**
- int **Norm3**
- int **BG**
- short **Shift**
- string **Sample**
- string **Comment**
- int **LightSource**
- int **Set**
- int **Tag**
- short **Grain**
- float **LightPower**
- short **SystemID**
- char **RESERVED_3_1** [36]
- float **OnTime**
- float **OffTime**
- int **EnableFlags**
- int **OnGateDelay**
- int **OffGateDelay**
- char **RESERVED_3_2**
- char **RESERVED_4_1** [20]
- int **CurveNo**
- int **TimeTick**
- int **StimPeriod**
- int **GateEnabled**
- int **GateStart**
- int **GateEnd**
- bool **PTenabled**
- char **RESERVED_4_2** [10]
- vector< int > **DPoints**
- int **mDateTime**
- int **IRR_DateTime**
- struct tm **DateTime**
- double **fadParameters** [2]
- double **eFadParameters** [2]
- double **fad_cov_0_1**
- gsl_matrix * **fadCovariantMatrix**
- FadFunction **fadFunction**
- double **rangeSignal** [2]
- double **rangeBackground** [2]

Static Public Attributes

- static const int **TIMEDATE_SIZE** =7
- static const int **USER_SEQUENCE_SIZE** =9
- static const int **SAMPLE_SIZE** =21
- static const int **COMMENT_SIZE** =81
- static const vector< string > **LTYPE**
- static const vector< string > **DTYPE**
- static const vector< string > **LIGHT_SOURCE**
- static const vector< string > **IRR_TYPE**
- static const vector< string > **IRR_UNIT**
- static const vector< string > **BL_UNIT**
- static const vector< string > **FAD_FORMULA**
- static const vector< string > **FAD_FUNCTIONS**
- static const vector< string > **VAR_NAMES**
- static const vector< string > **USER_VAR_NAMES**

Friends

- ostream & **operator**<< (ostream &stream, const Data &aData)
- ostream & **operator**<< (ostream &stream, const Data ∗aData)

The documentation for this class was generated from the following file:

- /home/boris/Dropbox/dokumenty/SURO/EURADOS_school/intercomp/my_program/qt_application/alive/src/data.-h

6.4 DataHandle Class Reference

Public Member Functions

- const vector< vector< int > > & **getDataPoints** ()
- const vector< Data > & **getData** ()
- int **getConfigurationData** (const string &confDataName)
- const double ∗ **getFadParameters** (const int which) const
- const double ∗ **getEFadParameters** (const int which) const
- void **setFadParameters** (const double ∗parameters, gsl_matrix ∗fadCovariantMatrix, const FadFunction fad-Function, const int which)
- void **setData** (const vector< Data > data)
- void **setDataNotPoints** (const int which, const Data &data)
- void **setRangeSignalBackground** (const double ∗signal, const double ∗background, const int which)
- set< int > **getWhichDType** (const string &whichType)
- BinReader ∗ **getBinReader** ()
- void **createData** (string name)
- void **createDb** (const QString dbName, const QString userName="root", const QString hostName="localhost", const QString password="")
- pair< double, double > **computeGeneralDose** (const pair< double, double > &signal, Function function, const double resolution=1e-1)
- pair< double, double > **computeFadCorr** (const int which, const FadFunction fadFunc) const
- double **getIntegral** (const int which, const double ∗integralRange) const
- std::pair< double, double > **getSignal** (const int which, const double ∗signalRange=NULL, const double ∗backgroundRange=NULL)
- void **setMainWindowData** (MainWindowData &data)
- const double ∗ **computeCalibration** (const double ∗signalRange=NULL, const double ∗background-Range=NULL, Function function=Linear)
- const double ∗ **chiSquareComputeGSL** (const vector< pair< pair< double, double >, pair< double, double > > > &inputData, Function function)
- const double ∗ **getParameters** () const
- const gsl_matrix ∗ **getCovariantMatrix** () const
- QSqlDatabase & **getDb** ()
- const MainWindowData & **getMainWindowData** ()
- double **getDoseUnc** (const double signal, const double ∗calibrationParameters) const
- const double ∗ **getInitialParameters** () const
- void **setInitialParameters** (const double ∗initialParameters)
- const vector< vector< int > > & **getValues** () const
- void **setValues** (vector< vector< int > > &values)
- void **setUsedMeasurements** (const set< int > measurements)
- const set< int > & **getUsedMeasurements** ()
- void **setFadCorrection** (const pair< double, double > &fadCorrection)
- const pair< double, double > & **getFadCorrection** () const
- void **setTimeUnit** (const TimeUnit timeUnit)

- TimeUnit **getTimeUnit** () const
- void **setFunction** (const Function function)
- Function **getFunction** () const
- void **setDatabasePath** (const string &databasePath)
- const string & **getDatabasePath** () const
- void **setDatabaseName** (const string &databaseName)
- const string & **getDatabaseName** () const
- pair< double, double > **getDoseCorrected** (const pair< double, double > &doseUnc, const pair< double, double > fadCorrection=pair< double, double >(0, 0))

Static Public Member Functions

- static DataHandle ∗ **getInstance** ()
- static TOOLS::func **getFunction** (Function function)
- static TOOLS::fadFunc **getFadFunction** (FadFunction function)
- static TOOLS::eFadFunc **getEFadFunction** (EFadFunction function)

Static Public Attributes

- static const vector< string > **FIT_FUNCTIONS**
- static const vector< string > **FIT_FORMULA**
- static bool **IS_NEW**

Friends

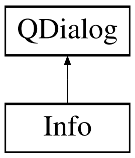
- ostream & **operator**<< (ostream &stream, const DataHandle &dataHandle)
- ostream & **operator**<< (ostream &stream, const DataHandle ∗dataHandle)

The documentation for this class was generated from the following file:

- /home/boris/Dropbox/dokumenty/SURO/EURADOS_school/intercomp/my_program/qt_application/alive/src/datahandle.-h

6.5 Info Class Reference

Inheritance diagram for Info:



Public Slots

- virtual void **accept** ()

Public Member Functions

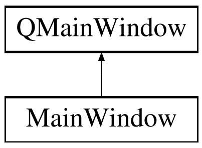
- **Info** (QWidget ∗parent=0)
- void **setUIData** (int which)
- void **storeData** ()

The documentation for this class was generated from the following file:

- /home/boris/Dropbox/dokumenty/SURO/EURADOS_school/intercomp/my_program/qt_application/alive/src/info.-
h

6.6 MainWindow Class Reference

Inheritance diagram for MainWindow:



Signals

- void **savedDbMeasurement** (int ID)

Public Member Functions

- **MainWindow** (QWidget ∗parent=0)
- void **showUIData** ()
- void **showTable** ()
- void **showPlot** ()
- void **savePlot** ()

Static Public Attributes

- static QString **TITLE**

The documentation for this class was generated from the following file:

- /home/boris/Dropbox/dokumenty/SURO/EURADOS_school/intercomp/my_program/qt_application/alive/src/mainwindow.-
h

6.7 MainWindowData Struct Reference

Data Fields

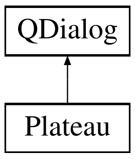
- string **name**
- string **dbName**
- string **user**
- double **IRR_Power**

The documentation for this struct was generated from the following file:

- /home/boris/Dropbox/dokumenty/SURO/EURADOS_school/intercomp/my_program/qt_application/alive/src/datahandle.-h

6.8 Plateau Class Reference

Inheritance diagram for Plateau:



Public Member Functions

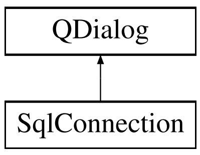
- **Plateau** (QWidget ∗parent=0)
- const Ui::Plateau ∗ **getUi** ()
- void **savePlateauPlot** ()

The documentation for this class was generated from the following file:

- /home/boris/Dropbox/dokumenty/SURO/EURADOS_school/intercomp/my_program/qt_application/alive/src/plateau.-h

6.9 SqlConnection Class Reference

Inheritance diagram for SqlConnection:



Public Slots

- void **updateTableFromCommand** ()
- void **showTable** (const QString &t)
- void **on_actionFetchDb_triggered** ()
- void **on_actionInsertRow_triggered** ()
- void **on_actionDeleteRow_triggered** ()
- void **currentChanged** ()

Signals

- void **statusMessage** (const QString &message)

Public Member Functions

- **SqlConnection** (QWidget ∗parent=0)
- bool **isOpenDb** ()
- virtual void **accept** ()
- const QTableView ∗ **getInfo1Table** () const
- int **getSqlEntry** (const int ID)
- void **insertRow** ()
- void **deleteRow** ()
- void **updateActions** ()
- void **showDbTable** ()
- void **onlyForSave** ()
- void **onlyForOpen** ()

Static Public Attributes

- static bool **IS_SAVE**

The documentation for this class was generated from the following file:

- /home/boris/Dropbox/dokumenty/SURO/EURADOS_school/intercomp/my_program/qt_application/alive/src/sqlconnection.-
h

6.10 SqlHandle Class Reference

Public Member Functions

- void **createMainTables** ()
- void **createSecondTable** (const int ID)
- void **insertIntoMainInfoTable** (const MainWindowData &mainWindowData)
- void **deleteMeasurement** (const int ID)
- const vector< Data > **getSqlData** (const int ID)
- Data **getDataFromQuery** (const QSqlQuery &query, const QSqlRecord &record)

Static Public Member Functions

- static SqlHandle ∗ **getInstance** ()
- static void **insertIntoSecondTable** (const int ID, const int measurement)
- static void **insertIntoThirdTable** (const int ID, const int measurement)

The documentation for this class was generated from the following file:

- /home/boris/Dropbox/dokumenty/SURO/EURADOS_school/intercomp/my_program/qt_application/alive/src/sqlhandle.-
h

Index