

# ASSIGNMENT 1 REPORT

GROUP 7

CSCI222

## CONTENTS

Overview.....	1
Group.....	1
Program Presentation .....	2
LIST OF FEATURES .....	2
Usage.....	3
Functional Requirements.....	9
Deliverables of Members.....	10
Thomas Nixon – TN941 .....	10
Ivana Ozakovic – io447.....	10
Philip Edwards – pme446.....	10
Josh Coleman – jjc224.....	11
Nicholas Morgan – nrm154.....	11
Phillip Mihajlovski – pm976 .....	11
Design & Implementation .....	12
FileArchiver .....	12
Database connection .....	13
Backend design .....	13
Creating a FileRecord .....	13
Creating a VersionRecord.....	14
Storing a file in the database. ....	14
Retrieving a file from the database.....	14
Storing multiple versions.....	14
Database .....	15
GUI Design Process .....	17
Design Diagrams .....	20
Use Case.....	20
Sequence Diagrams.....	21
Adding the initial file to the FileArchiver System.....	21
Updating a current file in the system.....	22
Retrieving A Current Version and Setting as Reference .....	23
Attempting To Save Without Having Made Changes.....	24
Viewing The Comment Of A File .....	25
Backend Class Diagram .....	26
Database Design – Entity Relationship Diagram.....	26
Design Changes.....	27
Design Iteration breakdown.....	27
Iteration 1 .....	27
Iteration 2 .....	28
Iteration 3 .....	29

Element List .....	29
Unit Testing Procedures .....	30
Version Control.....	30
Setup .....	31
Details/How we used it.....	31
Access and usage information .....	32
Group Records .....	35
Group Meeting Summary .....	35
Meeting One .....	35
Meeting Two .....	36
Meeting 3 .....	37
Meeting 4 .....	38
Meeting 5 .....	39
Group member work journal Samples .....	42
Sample of work diary from Thomas Nixon .....	42
Sample of work diary from Ivana Ozakovic .....	43
Supporting Code Samples.....	44
Support code for connecting to the database .....	44
Support for utilities for logging debug info.....	44
Unit Testing Samples .....	46
Testing for correctly appended path to file .....	46
Testing for retrieving filename functionality .....	46
Bug/Testing Log Samples .....	47
Appendices .....	48
Code Elements .....	48
Code Listing.....	49

## OVERVIEW

This project involves the design and implementation of a file management system. The system needs to allow users to store changes to files for later retrieval while remaining efficient in terms of storage requirements and processing speed. This system aimed to be a solution for arbitrarily large binary files, which is currently efficiently solved by most popular source code control versioning systems like Git.

The program's main functionalities are storing and managing files in persistent storage, as well as compressing and decompression of those files, keeping track of additions and removals and implementing a functional graphical user interface, the project uses a MySQL database as its backbone for data storage.

## GROUP

Role	Assignee
Manager (1)	Nicholas Morgan
Lead Designer (1)	Josh Coleman
Lead Implementer (1)	Philip Edwards
Designer (*)	many
Data Persistence Specialist (1)	Thomas Nixon
Systems integration and systems test (*)	Ivana Ozakovic, Philip Edwards
Documentation (*)	many
Implementer (*)	Philip Edwards, Thomas Nixon, Ivana Ozakovic, Josh Coleman
Document Backup Maintainer (1)	Thomas Nixon

## PROGRAM PRESENTATION

### LIST OF FEATURES

#### 1. Adding files to persistent storage

The system stores a copy of the file and its details in the persistent storage with a comment supplied by the user. The program compresses the given file/data to minimize storage space.

#### 2. Save modified version of the file

The program will detect if any changes are made in the current file and if a user attempts to save changes, a user will be asked to supply a comment to go with the new version of the current file. Only changes that were modified since the previous version archived will be saved in order to use the storage space efficiently.

#### 3. Display summary details of versions in storage for selected file

In case a user selects the file that is already in the storage, all versions of the file will be displayed in the table view with summary details of each version, which includes version number, date and time when version was created, and its size.

#### 4. Retrieve chosen version of the file

If the user wishes to retrieve a specific version of the file, the program retrieves the chosen version in the directory specified by the user along with the filename.

#### 5. Discard old

The program allows the user to set a “reference version”, which is used to purge all versions prior to the selected reference version.

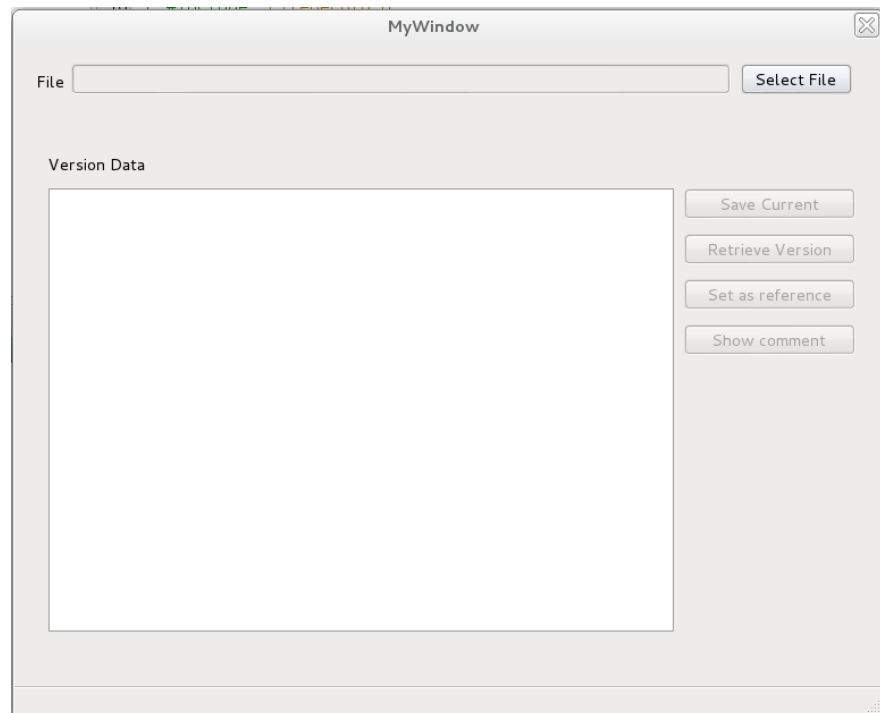
#### 6. Show comment associated with the version

The comment related to the selected version of the file from the table view will be displayed by clicking on the “Show Comment” button.

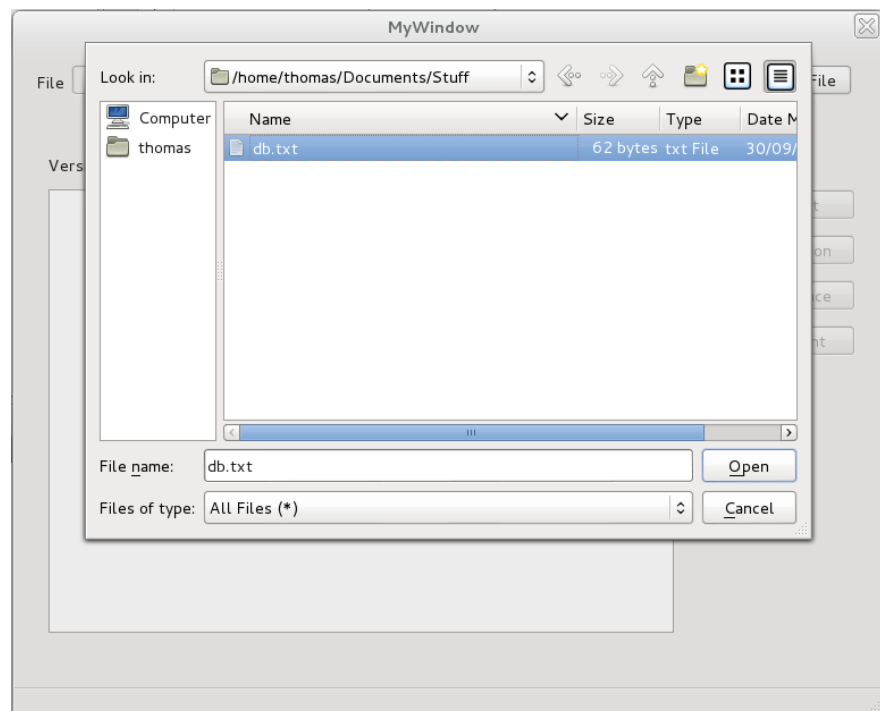
---

## USAGE

The initial main window gives user an option to select file with “Select File” button.



After clicking on “Select File”, the choose file window will appear for the user to select a file.



If the file is already saved in the persistent storage, stored versions of this file will be shown in the table.

MyWindow

File

Version Data

	Version #	Date	Size
1	1	2015-10-01 09:22:13	62
2	2	2015-10-01 09:22:13	72
3	3	2015-10-01 09:22:13	70

Otherwise, the program will create the initial version in the database and ask user for a comment to go with the initial file version.

MyWindow

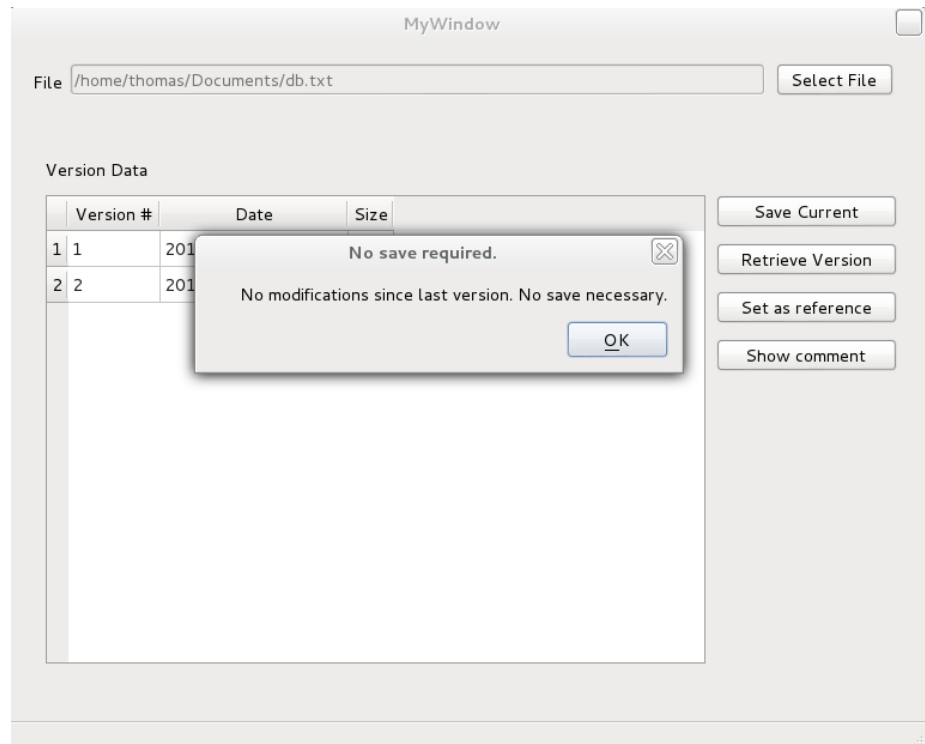
File

Version Data

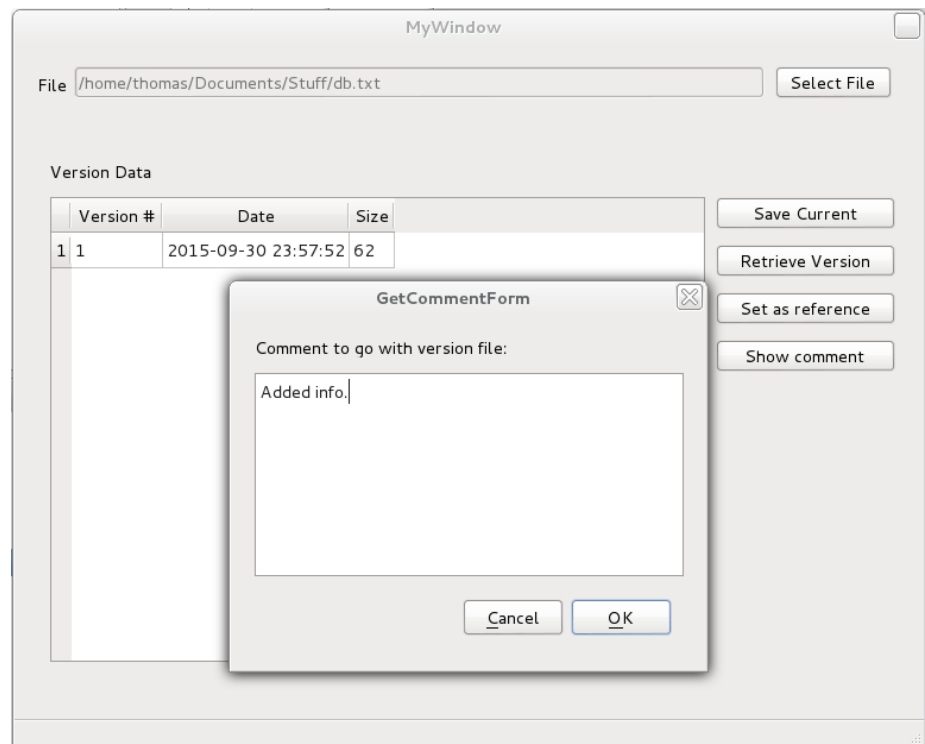
GetCommentForm

Comment to go with version file:

If the user attempts to save the current version of the file that has not been changed, the message box will show up to inform the user that there is no need for saving the current version.

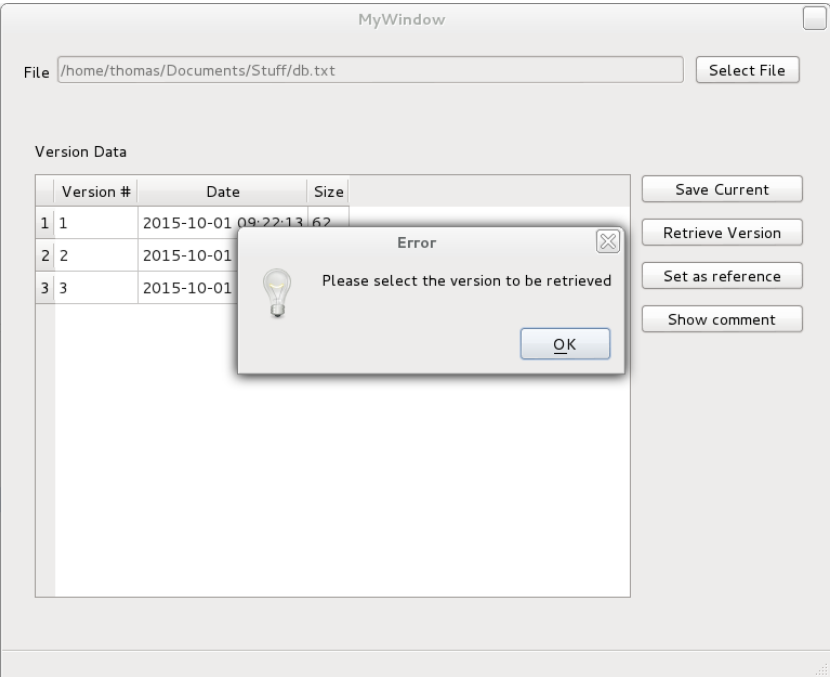


In the case a change in the current version was detected, the user will be asked to supply a comment for the new version and the new version of the file will be displayed in the table.

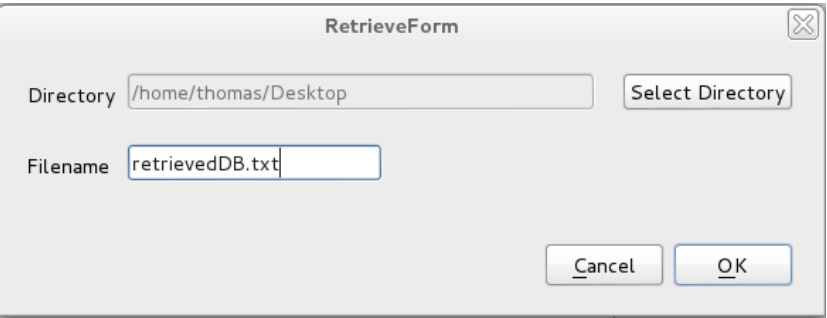




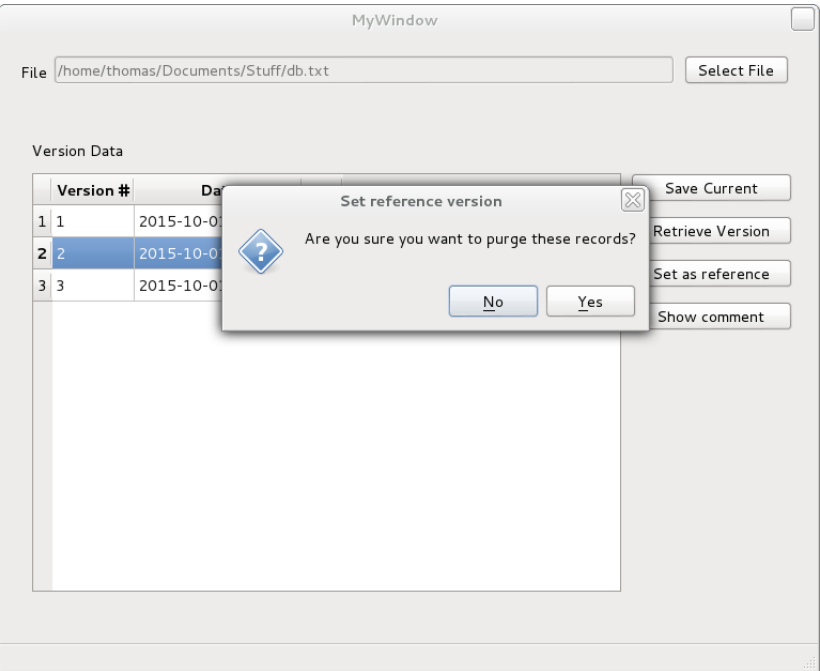
The retrieve version button provides the user the ability to retrieve the version of the file that is selected in the table view. If the file version to be retrieved was not selected in the table, user will be informed.



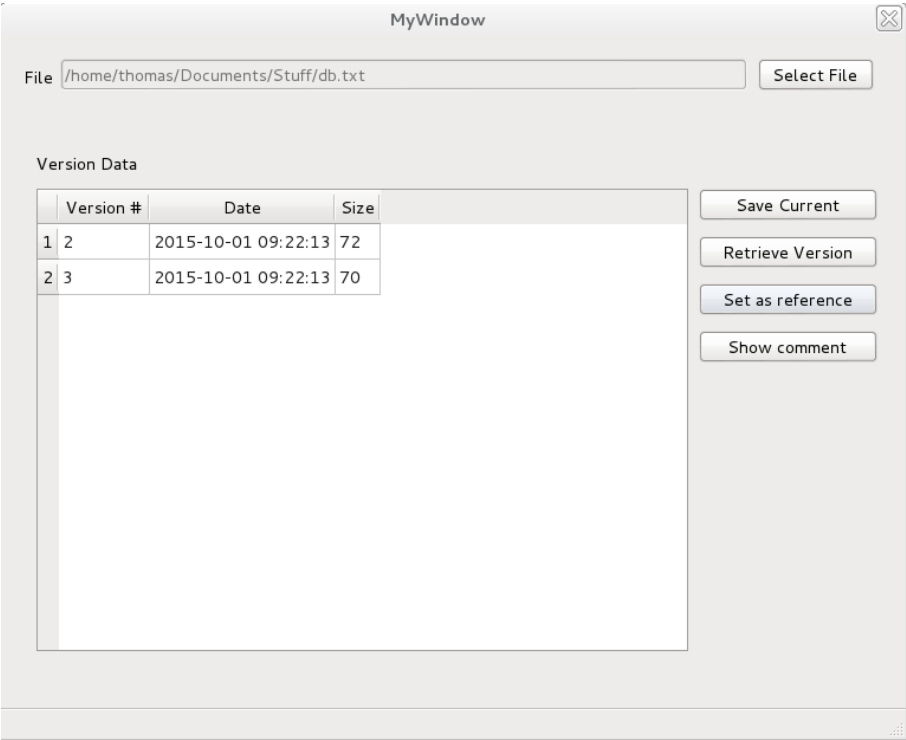
Otherwise, the retrieve form dialog will show up asking user to specify the directory where the retrieved version will be saved, along with the filename.



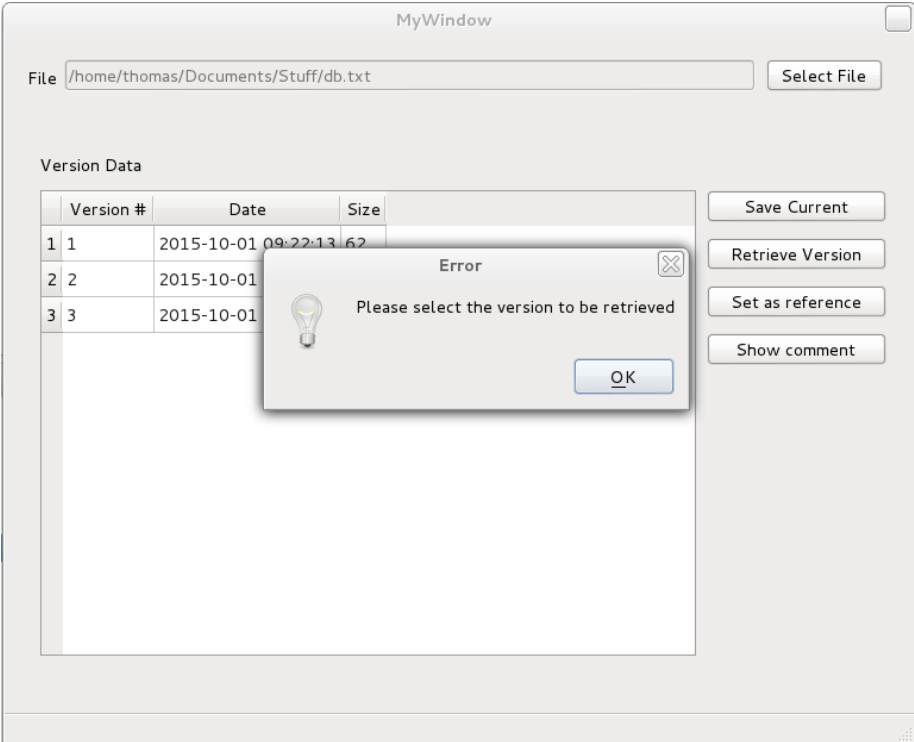
Set as reference option allows user to select version that will be used as a reference to delete all previous versions of the file that are not needed anymore. The program will ask for confirmation to proceed with deletion.



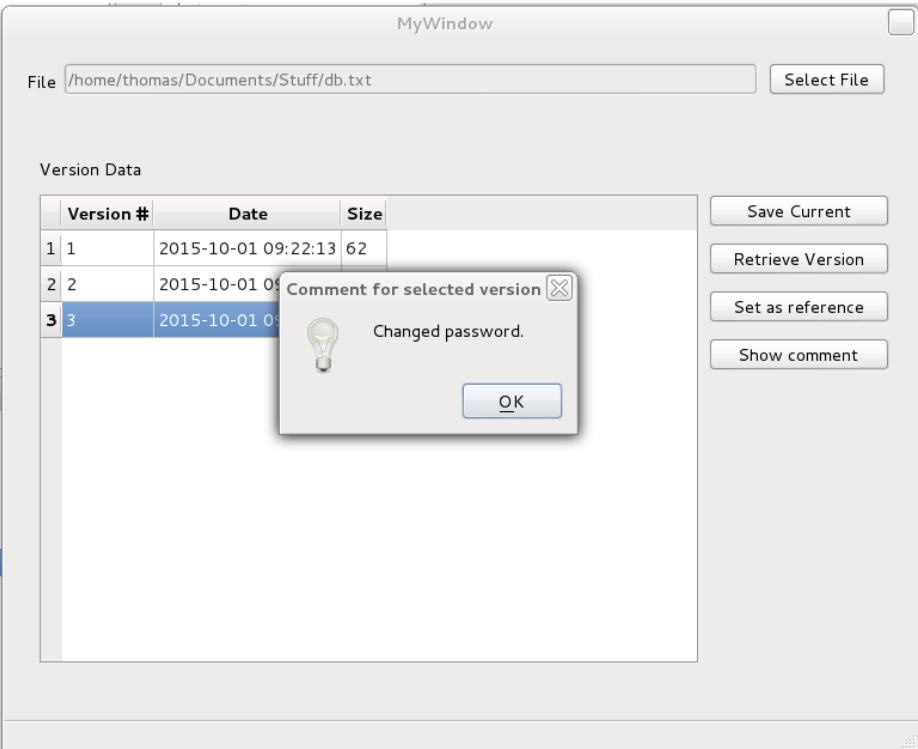
After a purge has completed, the program will then show only the available versions



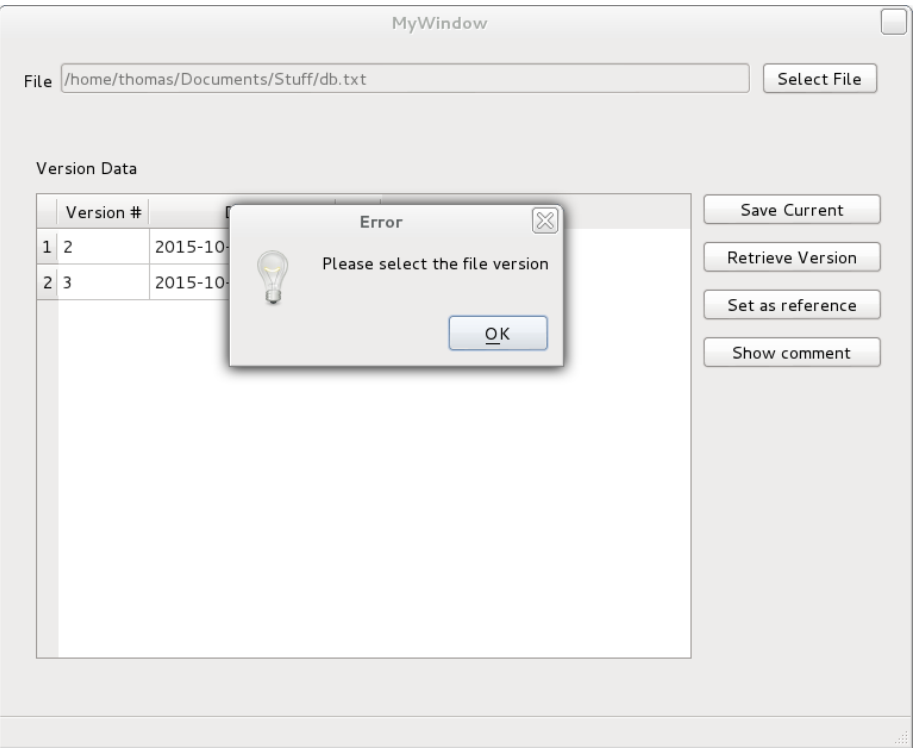
When attempting to retrieve a version when the reference version is not selected in the table, the program will notify the user.



If a user wishes to view the comment related to specific version, the show comment button will invoke the information dialog with the comment of the selected version.



In case the user did not select a version, the program will notify the user.



## FUNCTIONAL REQUIREMENTS

The program specification had 9 functional requirements that needed to be met.

ID	TITLE	PRIORITY	STATUS
1	Create Initial Archive	1	Completed
2	Detect Changes	2	Completed
3	Save Modified Version	3	Completed
4	Display a summary of versions in storage	4	Completed
5	Retrieve Chosen Version	5	Completed
6	Show comment associated with version	6	Completed
7	Use Compression	7	Completed
8	Store incremental changes	8	Completed
9	Discard Old Versions	9	Completed

## DELIVERABLES OF MEMBERS

### THOMAS NIXON – TN941

- Researched the benefits of using MySQL
- Designed new database layout
- Created database diagrams
- Wrote SQL code for database
- Created the SQL statements and C++ code required for the program
- Created initial document in Google Docs for outlining responsibilities and requirements
- Defined layout of classes and member functions and their interaction (With Phil Edwards)
- Implemented VersionRecord member function code
- Moved documents from Google Docs to Git
- Wrote code to handle compression of files and implemented it in required functions

### IVANA OZAKOVIC – IO447

- Created GUI main window and dialogs in QtBuilder.
- Created all the GUI functionality in collaboration with Josh.
- Took GUI screenshots for the report.
- Created Program Presentation section for the report.
- Created GUI Implementation and Planning section for the report.
- Declared data members and functions in the VersionRecord.h.
- Created VersionRecord.cpp file and set up stubs for declared functions.
- CPPUnit Tests

### PHILIP EDWARDS – PME446

- |  |  |
|--|--|
| <ul style="list-style-type: none"><li>• Created GitHub suppository for project</li><li>• Created initial class files</li><li>• Created the Netbeans project</li><li>• Wrote coding standard document</li><li>• Fixed indentation issues across Netbeans project</li><li>• Moved database connection to a static function in a class</li><li>• Add dependency for MySQL to Netbeans project</li><li>• Fixed header guard bug</li><li>• Added murmur hash to implementation</li><li>• Implemented FileRecord</li><li>• Implemented FileArchiver</li><li>• Implemented RetrieveVersionRecord in VersionRecord</li></ul> | <ul style="list-style-type: none"><li>• Fixed Bugs VersionRecord</li><li>• Created Wiki for Git</li><li>• Created TODO document</li><li>• Wrote Git Primer for other members to refer to</li><li>• Defined layout of classes and member functions and their interaction (With Thomas Nixon)</li><li>• Created a modified version of murmur hash function to read from file.</li><li>• Wrote test code for functions in FileArchiver &amp; FileRecord</li><li>• Added logging to program</li><li>• Made some modifications to database</li><li>• Created MySQL database server setup for other group members to use</li></ul> |
|--|--|

---

#### JOSH COLEMAN – JJC224

- Created all functionality for FileLib.
- Created all CppUnit tests for FileLib.
- Took images of FileLib code (both header/stubs and source/definitions).
- Took images of FileLib CppUnit tests.
- Created the table model for the table view in the GUI.
- Worked on all GUI table/button functionality (in collaboration with Ivana).
- Report on construction phase iterations

---

#### NICHOLAS MORGAN – NRM154

- Meeting Reports
  - Timekeeping and note taking
- Documentation
  - Elements and unit testing procedure
  - Supporting code samples and listings
  - Version Management – initial
  - Detailed meeting report – revision
  - Bug log/Testing sample section
- Project collation

---

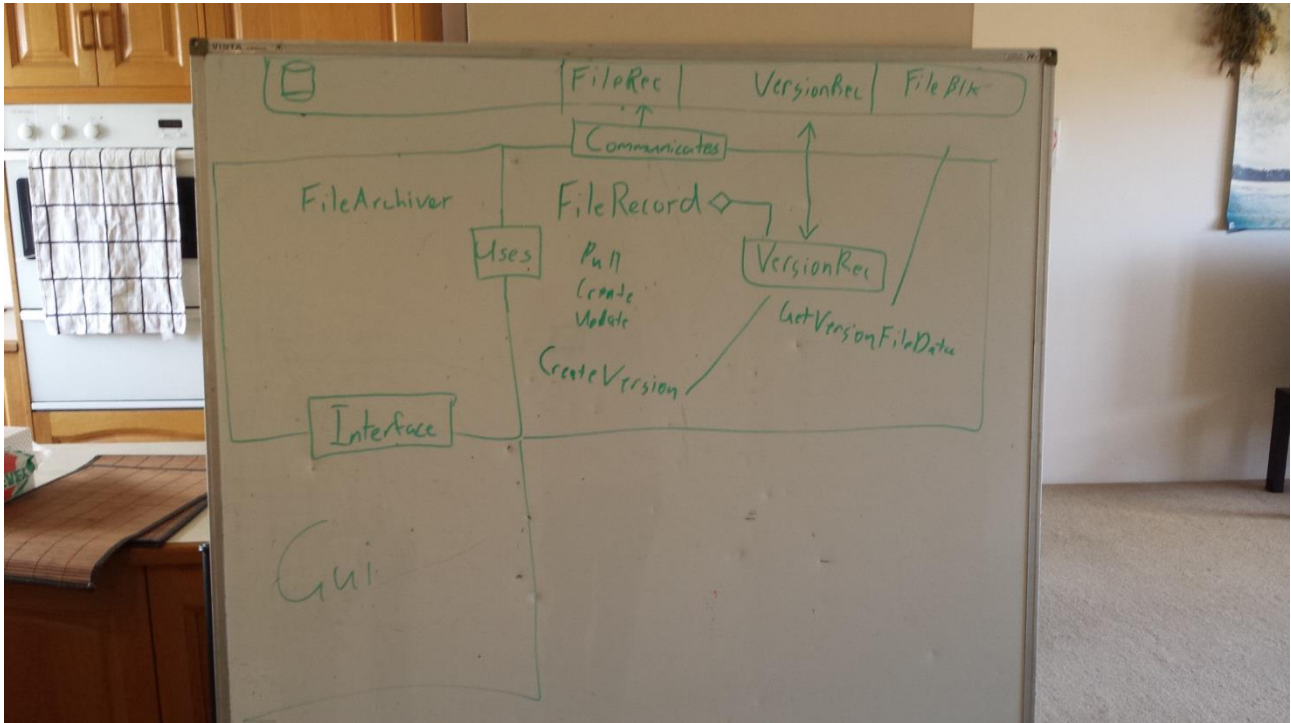
#### PHILLIP MIHAJLOVSKI – PM976

- Use Case and Sequence Diagrams
- Documentation – Version Management
- Documentation - Diagrams

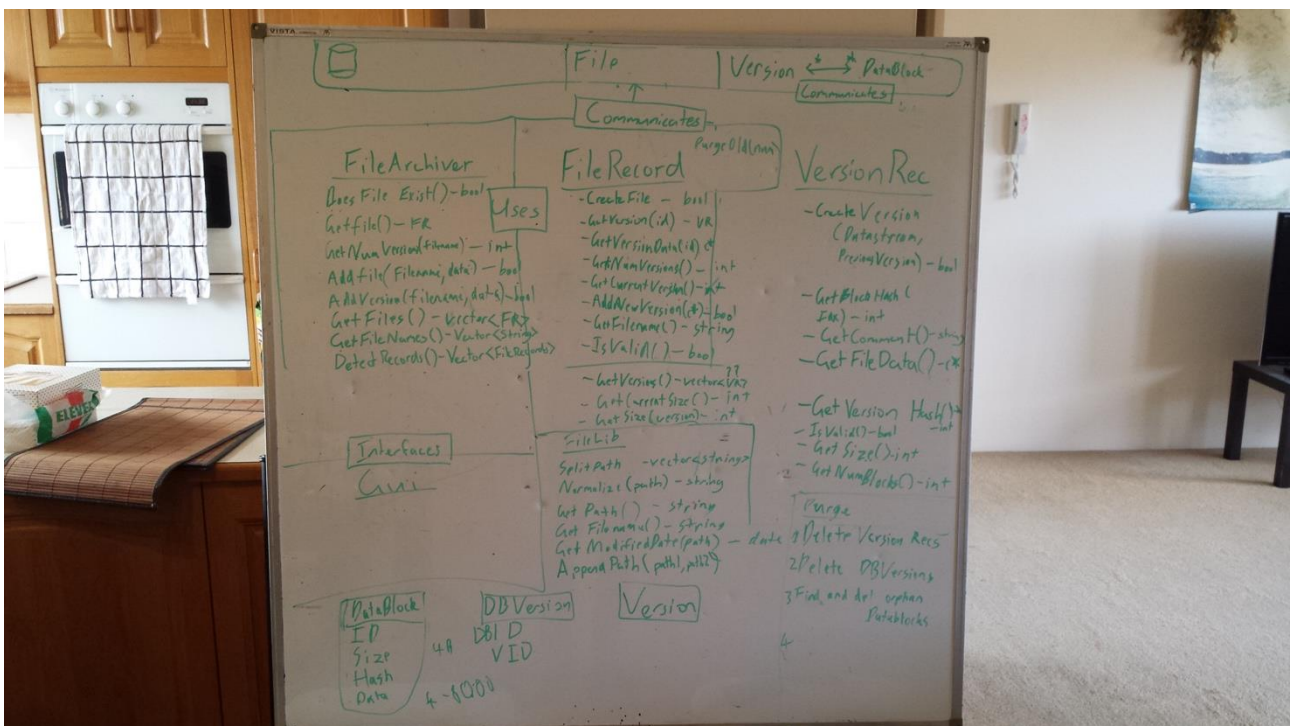
## DESIGN & IMPLEMENTATION

### FILEARCHIVER

To understand the program better we wrote the main functions on a white board



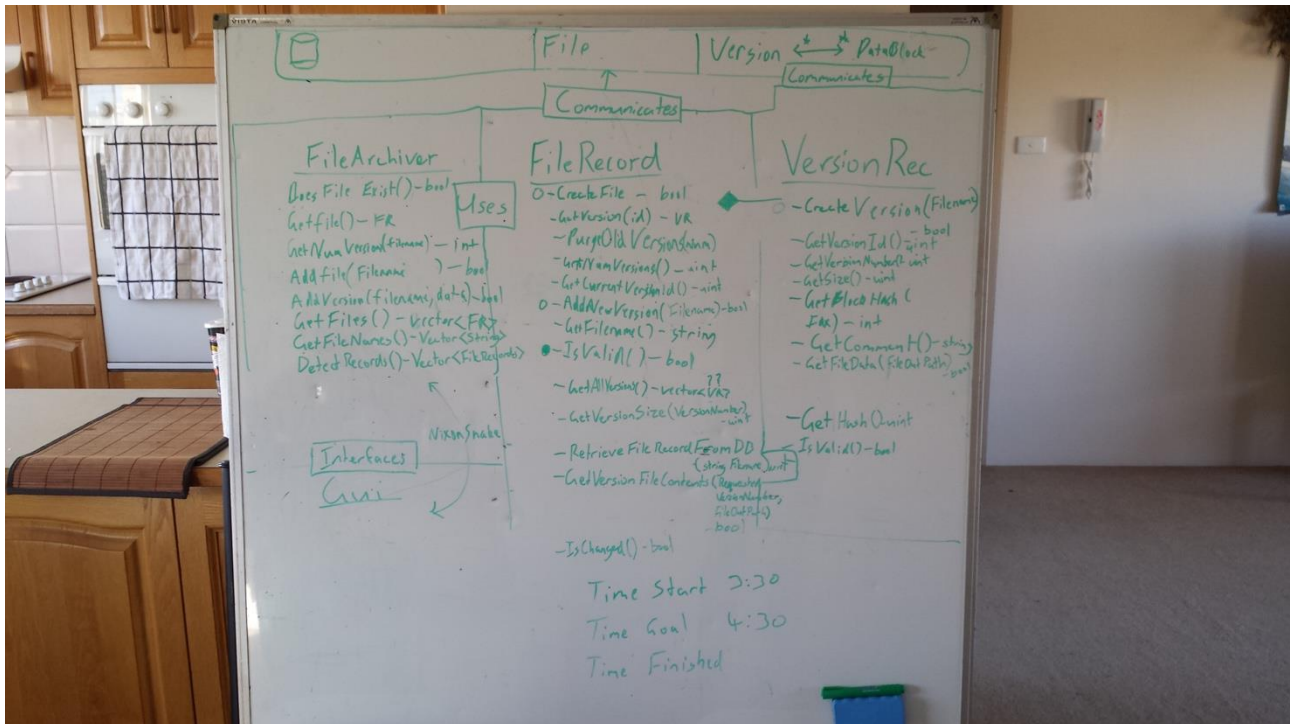
This gave a good starting point. From here we began to flesh it out and understand what functions were needed and how to communicate between the different components.



Expansion from the starting point, designed to provide, at a glance an overview of the structure of the

entire program, to provide insight into program flow and to highlight any area's where improvements could be made.

The further we got the better we understand what we were working towards. We were able to trim excess functions and remove anything that was doubled up.



All the class files were designed from this initial layout. From here we decided to get some of the core functionality going.

## DATABASE CONNECTION

Our program uses a single database connection to reduce load on the database if multiple clients are connecting to the same version database. We have a global function which manages the database connection and passes a pointer to any part of the program that requires access to the database.

## BACKEND DESIGN

The design of the backend was a collaborative effort primarily between the Lead Implementer and the Data Persistence Specialist with input from the implementers of the GUI. During this design phase the database schema, the backend API, and the associated classes were designed. This enabled all incumbents to ensure that the correct data was stored, that there was a useful interface to the backend, and that all functional requirements were met.

## CREATING A FILERECORD

FileRecord is a class that should have a direct correlation with an entry in the database. It provides a function `IsValid()` which can be used to check that the FileRecord is safe to use. If this is true we can use it to pull information from the database or to commit new versions.



FileRecord provides easy access to get a record from the database through the constructor, as well as providing an interface for creating a new FileRecord in the database.

## CREATING A VERSIONRECORD

A VersionRecord is created through a valid FileRecord object. A new version record should never be created in other places in the program. Getting a reference to an existing file record can be accomplished with the VersionRecord constructor in a similar way to using filerecord, however filerecord also provides functions for retrieving a VersionRecord.

## STORING A FILE IN THE DATABASE.

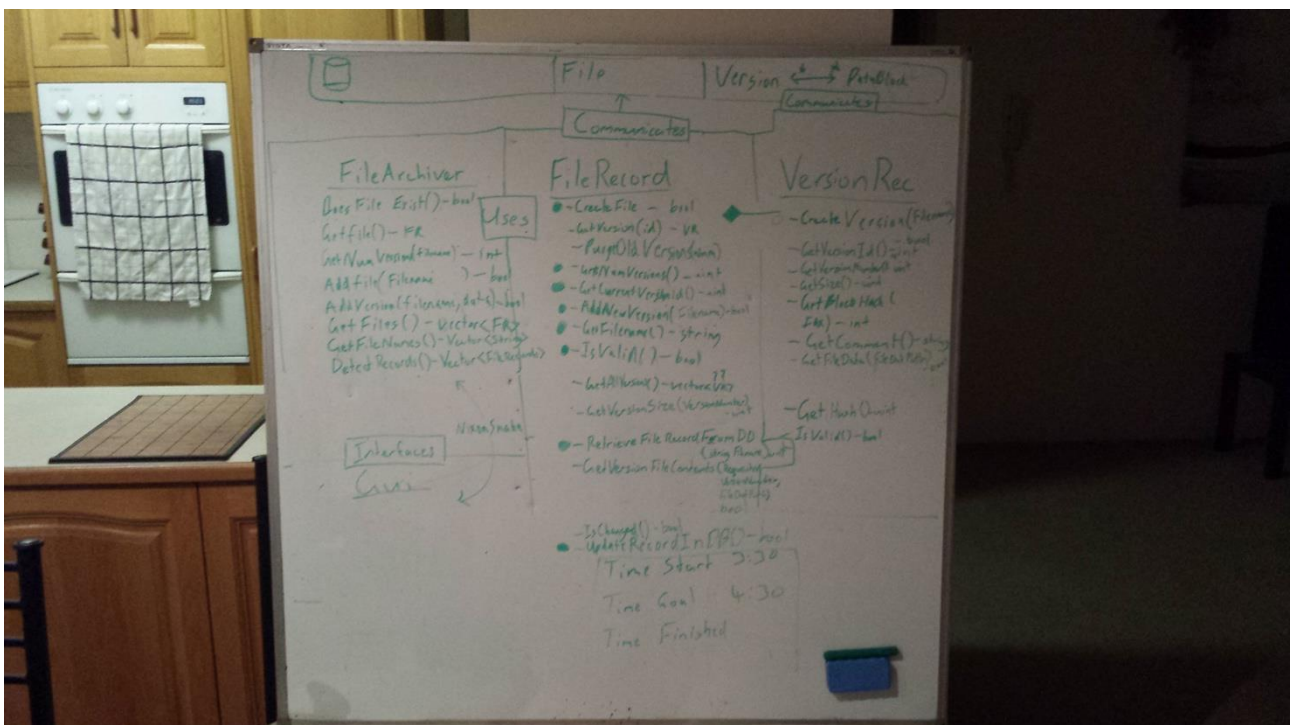
Files are stored in the database when a new VersionRecord is created through FileRecord. The VersionRecord class handles creation of the VersionRecord database entry. It also handles checking for duplicate blocks and creation of VtoB records which associate blocks with a version record.

## RETRIEVING A FILE FROM THE DATABASE.

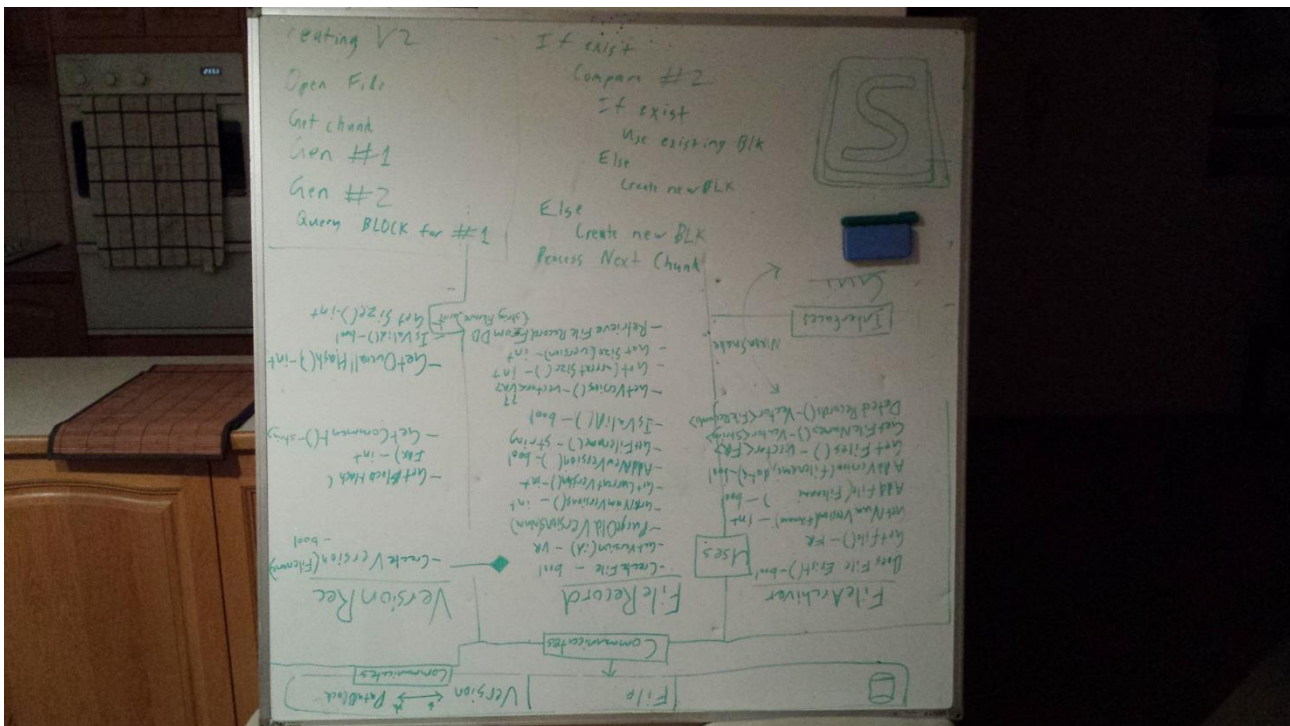
Retrieving a file from the database is accomplished through a valid VersionRecord instance. It will retrieve a list of VtoBs and then gather all required blocks and store them in a file on the database, before finally decompressing the file to the final destination.

## STORING MULTIPLE VERSIONS.

Multiple versions are handled easily by associating a VersionRecord with a FileRecord in the database. Actual file data is broken up into blocks of 4 or 8kb which are verified as unique. A block will never be stored in the database twice. A version then has 1 or more VtoB entries associated with it, which link a block with the VersionRecord in the correct order.

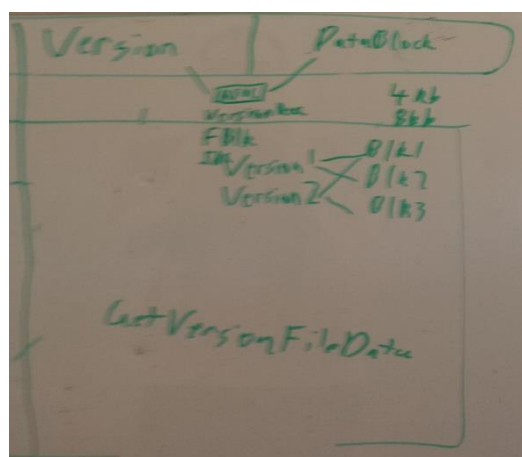


The whiteboard was also used to help us come up with logic in some of the functions. That way two people could look at it and make sure it made sense before implementing.

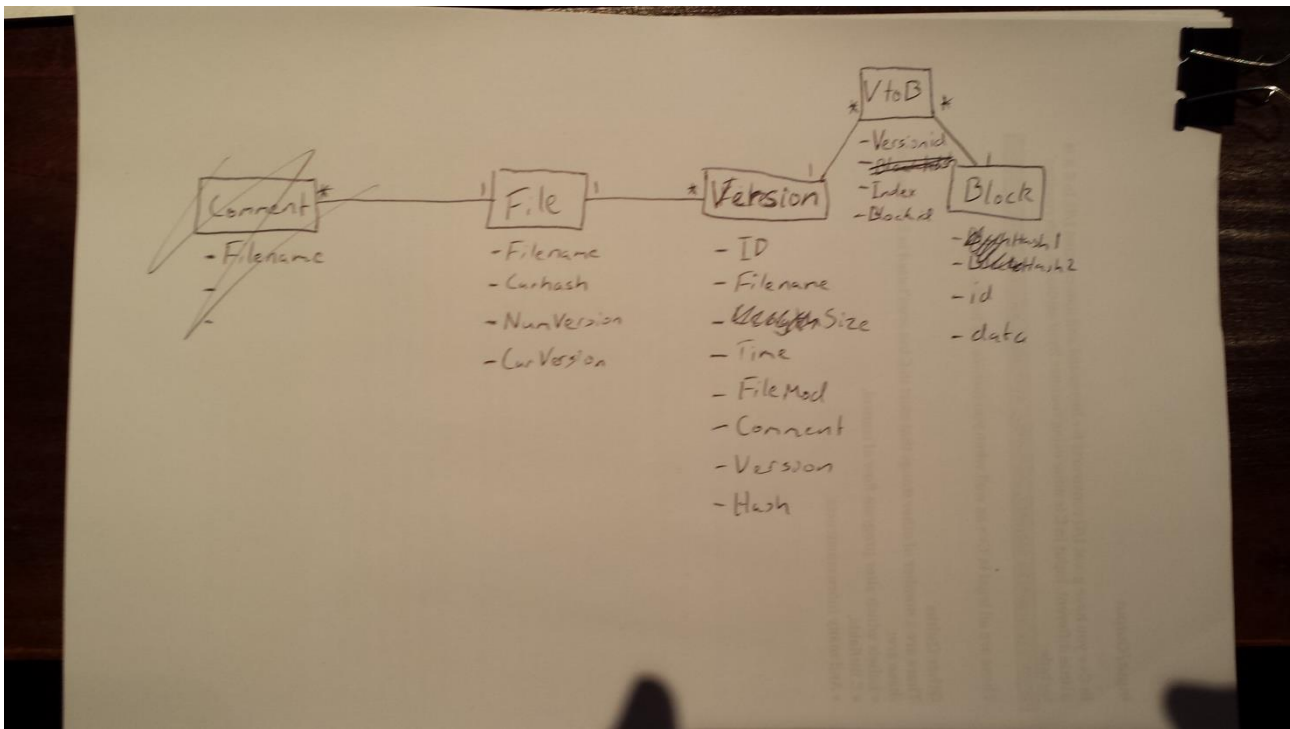


## DATABASE

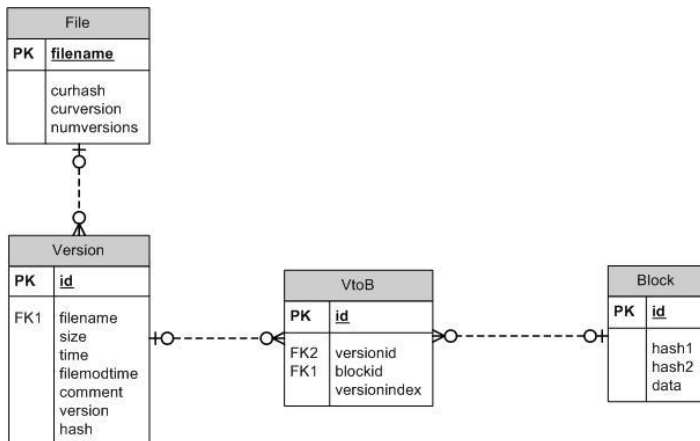
When looking at how to store the data of the file we decided to break the file down into blocks. These blocks would be linked to a version with the intermediate table VtoB which would keep track of the blocks required for a version file and the appropriate index. This allowed for a many to many relationships between Version and Block so that we could store blocks that were the same under different versions without needing to duplicate data.



After deciding on how to store the files we redesigned the database to better reflect what we wanted to achieve. Redundant tables and fields were removed. Names of tables were modified to better reflect the data within them and reduce confusion.



Finally, an ERD was created to show how the database worked. This was useful if anyone needed to refer to the database or see where data was stored in the database. The relationship between the tables is also shown. You can see that there are two hashes in the Block table. This was done reduce the number of collisions possible. This is very important in the Block table as a collision would destroy the integrity of our version files. When a hash1 is the same as a block that is stored it generates a second hash (hash2) which has a different seed. We found this a suitable solution. Below you can see the ERD and the first version of the database creation code in SQL.



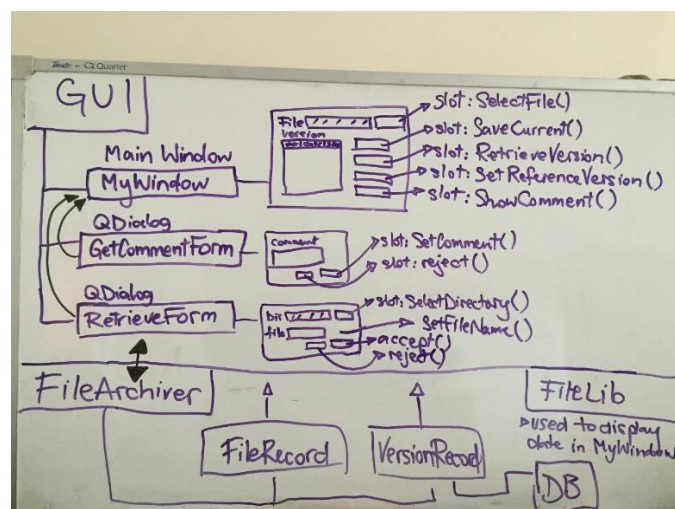
```

1 create table File (
2     filename varchar(767) not null,
3     curhash int(11) unsigned,
4     curversion int(11) unsigned,
5     numversions int(11) unsigned,
6     primary key (filename)
7 );
8
9 create table Version (
10     id int(11) unsigned not null auto_increment,
11     filename varchar(767),
12     size int(11) unsigned,
13     time int(11) unsigned,
14     filemtime int(11) unsigned,
15     comment mediumtext,
16     version int(11) unsigned,
17     hash int(11) unsigned,
18     primary key (id),
19     foreign key (filename) references File(filename)
20 );
21
22 create table Block (
23     id int(11) unsigned not null auto_increment,
24     hash1 int(11) unsigned,
25     hash2 int(11) unsigned,
26     data mediumblob,
27     primary key (id),
28     index(hash1),
29     index(hash2)
30 );
31
32 create table VtoB (
33     id int(11) not null auto_increment,
34     versionid int(11) unsigned,
35     blockid int(11) unsigned,
36     versionindex int(11) unsigned,
37     primary key (id),
38     foreign key (versionid) references Version(id),
39     foreign key (blockid) references Block(id)
40 );
41
42 alter table Version auto_increment = 1;
43 alter table Block auto_increment = 1;
44 alter table VtoB auto_increment = 1;
45
46 delete from VtoB;
47 delete from Block;
48 delete from Version;
49 delete from File;
50
51 drop table VtoB;
52 drop table Block;
53 drop table Version;
54 drop table File;
  
```

## GUI DESIGN PROCESS

In the process of getting the GUI started, the initial task was to create the main window of the program and its dialogs using Qt Designer, which generated all necessary files for the GUI elements in the NetBeans project.

Next thing on the agenda was to set up all needed slot functions and connect them to the elements of the GUI. After a discussion with other implementers to understand how different components of the program will interact with each other, diagram was drawn on the whiteboard to provide an overview of which elements will contain each functionality, and how it will interact with rest of the program.





The following image is the example of how the diagram shown was used to start off the MyWindow class.

```
21 MyWindow::MyWindow() {
22     widget.setupUi(this);
23
24     widget.saveCurrentBtn->setEnabled(false);
25     widget.retrieveVersionBtn->setEnabled(false);
26     widget.setReferenceBtn->setEnabled(false);
27     widget.showCommentBtn->setEnabled(false);
28
29     //connect SelectFile() with selectFileBtn
30     connect(widget.selectFileBtn, SIGNAL(clicked()), this, SLOT(SelectFile()));
31     //connect saveCurrentBtn with SaveCurrent()
32     connect(widget.saveCurrentBtn, SIGNAL(clicked()), this, SLOT(SaveCurrent()));
33     //connect RetrieveVersion() with retrieveVersionBtn which will open RetrieveForm
34     connect(widget.retrieveVersionBtn, SIGNAL(clicked()), this, SLOT(RetrieveVersion()));
35
36     //connect setReferenceBtn with SetReferenceVersion() to delete unnecessary file versions
37     connect(widget.setReferenceBtn, SIGNAL(clicked()), this, SLOT(SetReferenceVersion()));
38     //connect ShowComment() with showCommentBtn
39     connect(widget.showCommentBtn, SIGNAL(clicked()), this, SLOT(ShowComment()));
40 }
```

From the above whiteboard diagram it can be seen that the GUI will be communicating mostly with FileArchiver class, but also using FileRecord and VersionRecord class through FileArchiver.

This was a helpful way to start implementing the functionality of the slot functions. First slot function needed to be defined was SelectFile(), as all other functionality of the program is based on the file that the user has selected to initiate the file or retrieve version of the chosen file from the persistent storage.

As the backend code was initially tested separately from the GUI functionality, the implementation with the GUI was pretty straight forward, as the test code provided insight of the usage of different class objects and how they interact.

The most time was spent on the decision which model will be used to display retrieved file versions in the table view. In order to find an easier and more concise solution than writing our own QAbstractTableModel class, we have researched the abilities of QtCore classes and decided to use QAbstractItemModel class. QAbstractItemModel supplied all the functionality required for our program.

The following screenshot shows the usage of table view and QAbstractItemModel to retrieve information for a selected version of the file by wrapping the RetrieveVersionDataForFile function in MyWindow class.

```
QStandardItemModel *myModel = new QStandardItemModel(fileRec.GetNumberOfVersions(), 3, this);
myModel->clear();

myModel->setHorizontalHeaderItem(0, new QStandardItem(QString("Version #")));
myModel->setHorizontalHeaderItem(1, new QStandardItem(QString("Date")));
myModel->setHorizontalHeaderItem(2, new QStandardItem(QString("Size")));

vector<VersionRecord> versionRecs = fileRec.GetAllVersions();
unsigned int currentRow = 0;

for(vector<VersionRecord>::iterator it = versionRecs.begin(); it != versionRecs.end(); ++it)
{
    myModel->setItem(currentRow, 0, new QStandardItem(QString(boost::lexical_cast<string>(it->GetVersionNumber()).c_str())));
    myModel->setItem(currentRow, 1, new QStandardItem(QString(FileLib::GetFormattedModificationDate(fileRec.GetFilename()).c_str())));
    myModel->setItem(currentRow, 2, new QStandardItem(QString(boost::lexical_cast<string>(it->GetSize()).c_str())));

    ++currentRow;
}

widget.tableView->setModel(myModel);
widget.tableView->resizeColumnsToContents();
widget.tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);
widget.tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
widget.tableView->setSelectionMode(QAbstractItemView::SingleSelection);
```

After all functionality was implemented, we focused on making sure that the control flow and error-handling logic of the program solves each issue where users can make erroneous interactions through the GUI.

Each possible user interaction case was considered and we made sure to inform users how to achieve what was intended by supplying information in form of different kinds of message dialogs.

The screenshot below shows how this was implemented in the case where the user tries to retrieve a version of a file, but has not selected the version to be retrieved from the display in the table view.

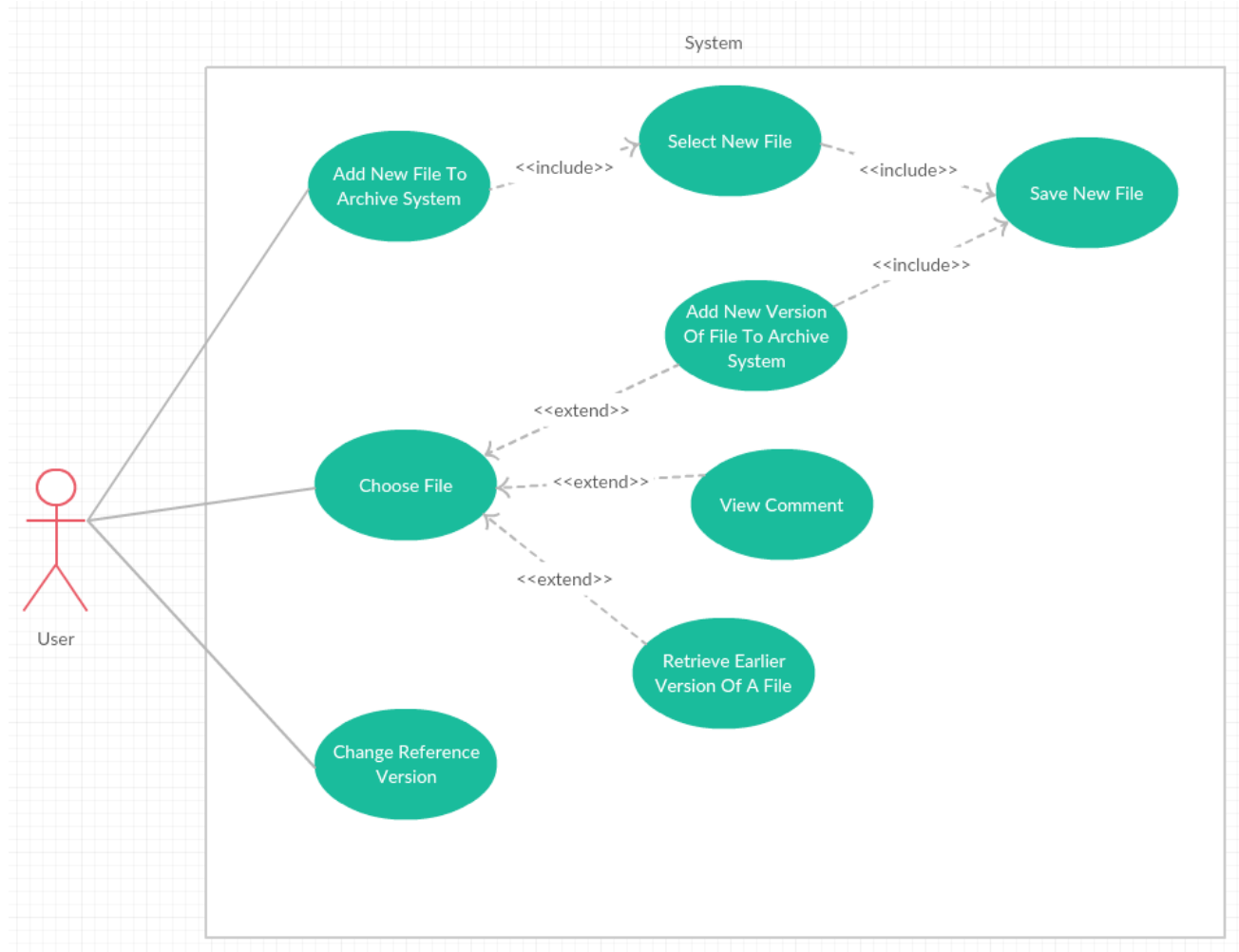
```
if(!(indexes.size() > 0))
{
    QMessageBox msgBox(QMessageBox::Information, "Error",
        "Please select the version to be retrieved", QMessageBox::Ok, 0);

    msgBox.exec();
    return;
}
```

Finally, after GUI was completely functional and there were no bugs, we made sure everyone was happy with the final product.

## USE CASE

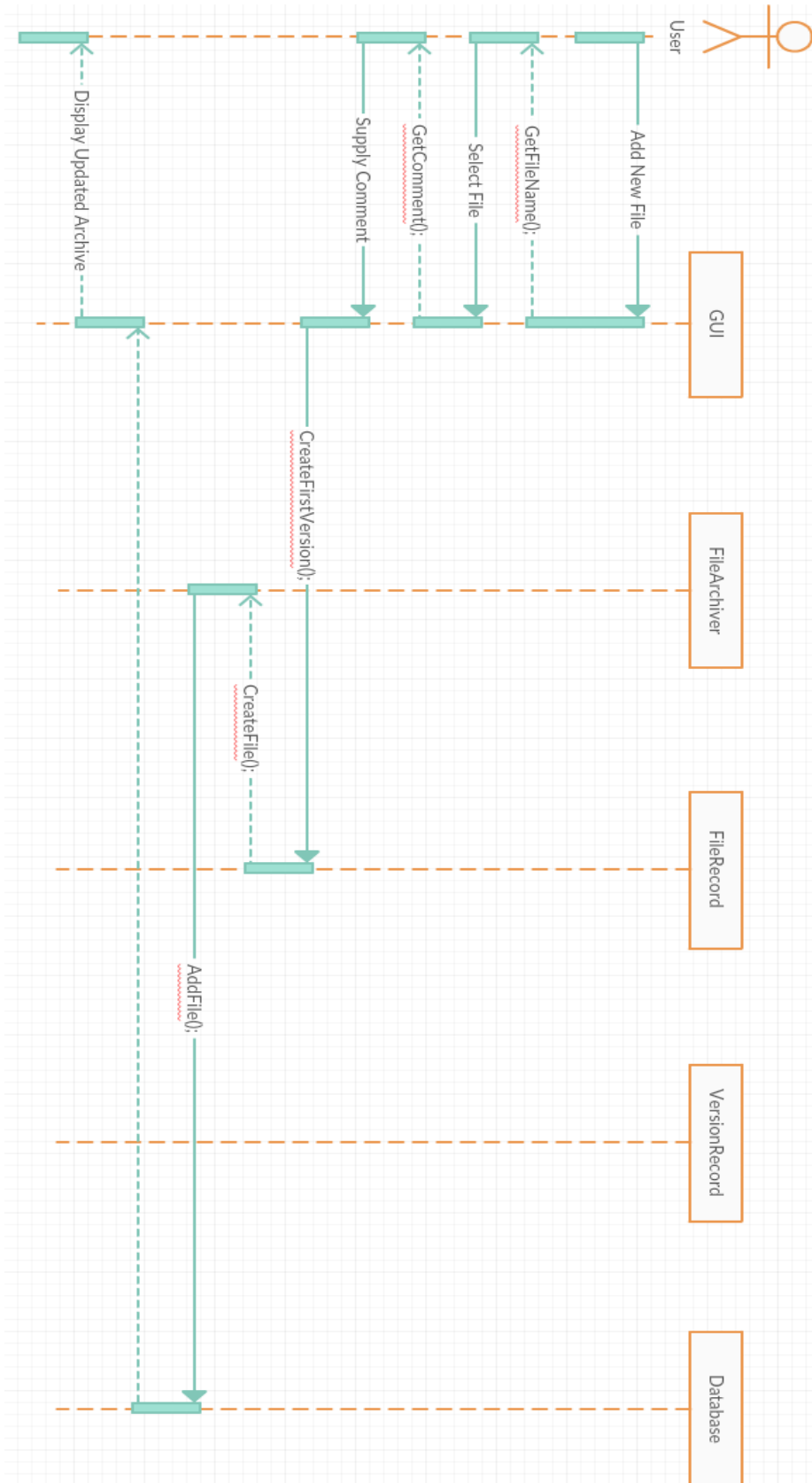
This use case includes the use cases U1 through to U5.



## SEQUENCE DIAGRAMS

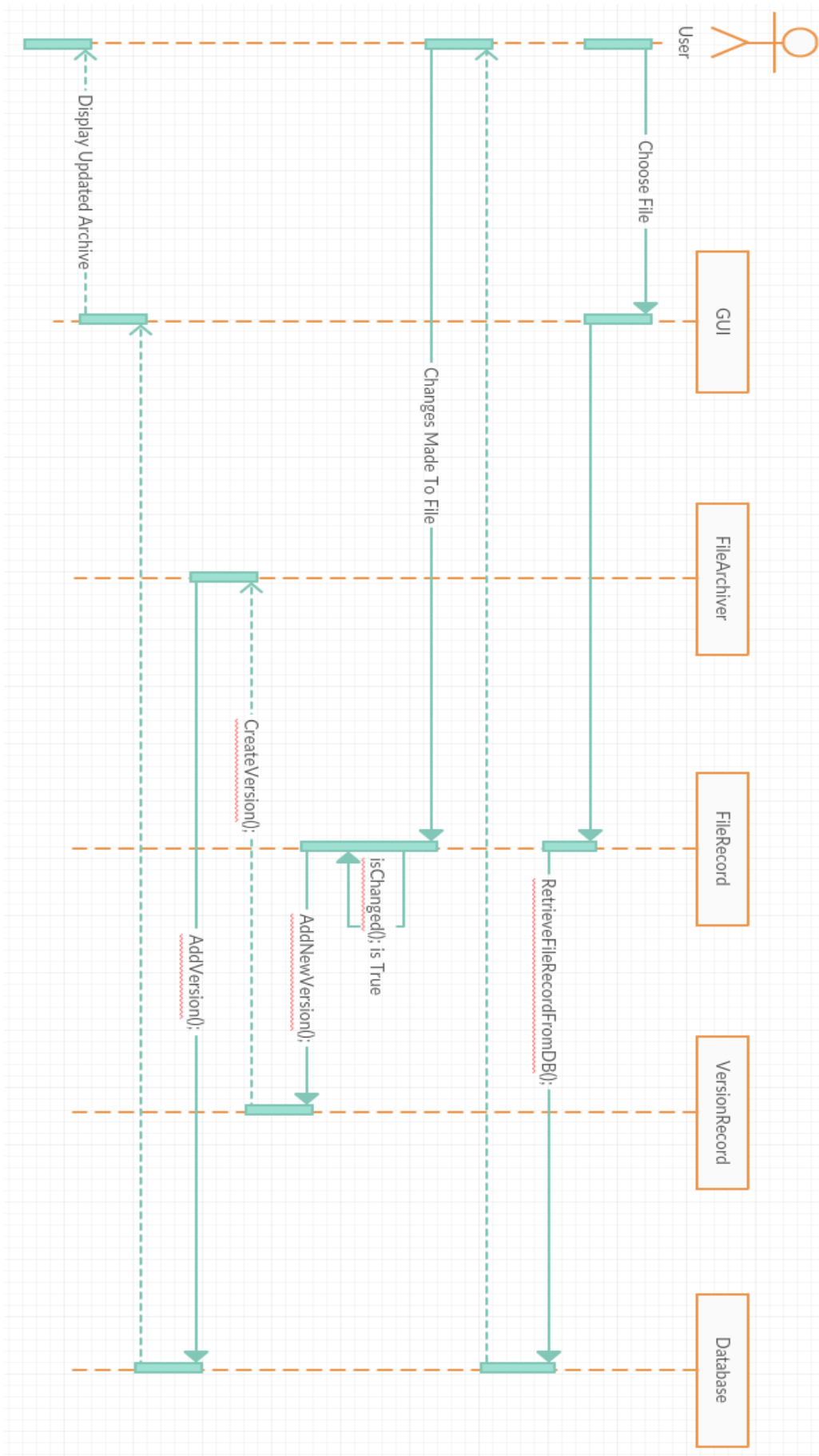
The sequence diagrams will highlight how the system uses its classes to work together.

### ADDING THE INITIAL FILE TO THE FILEARCHIVER SYSTEM.

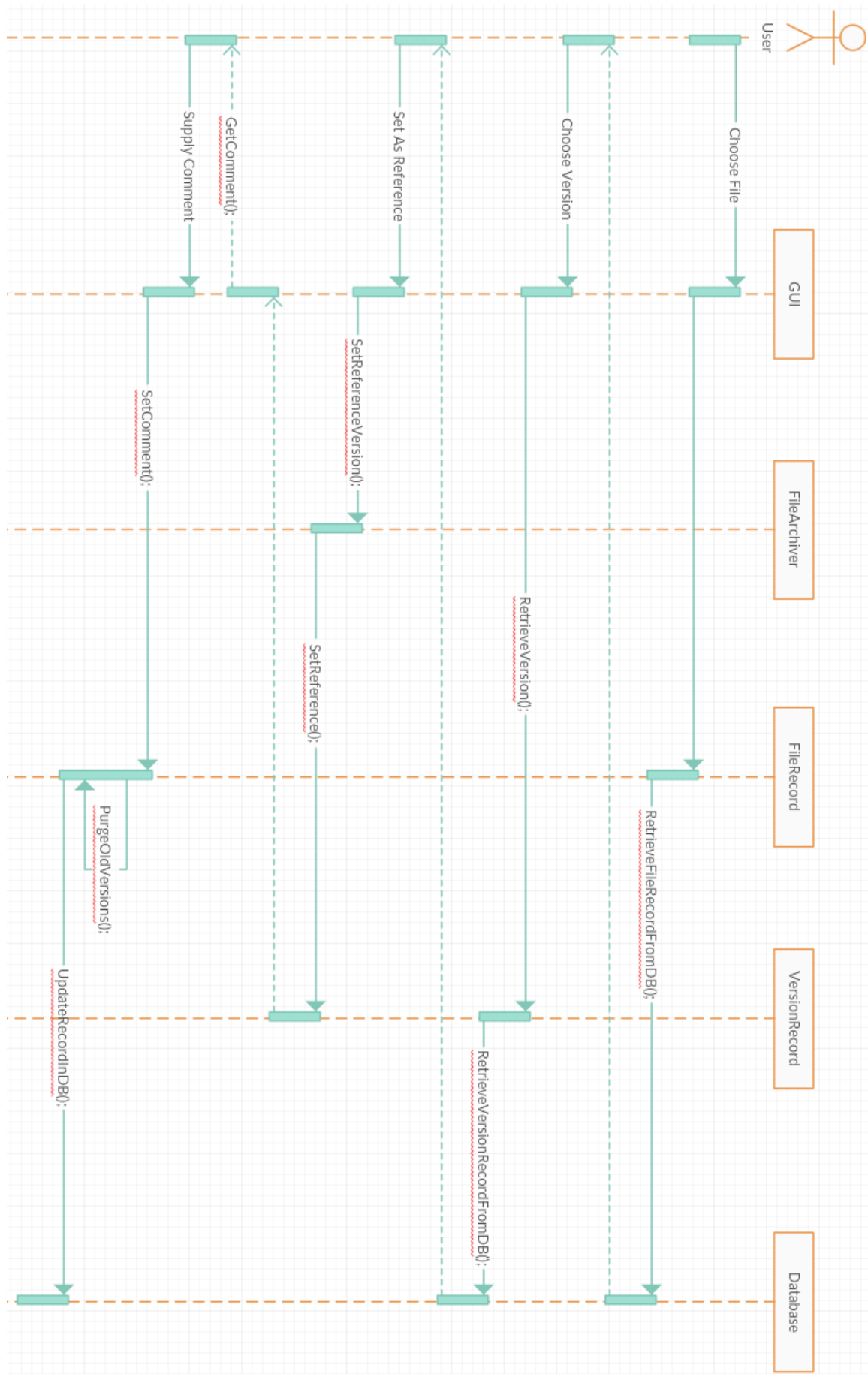




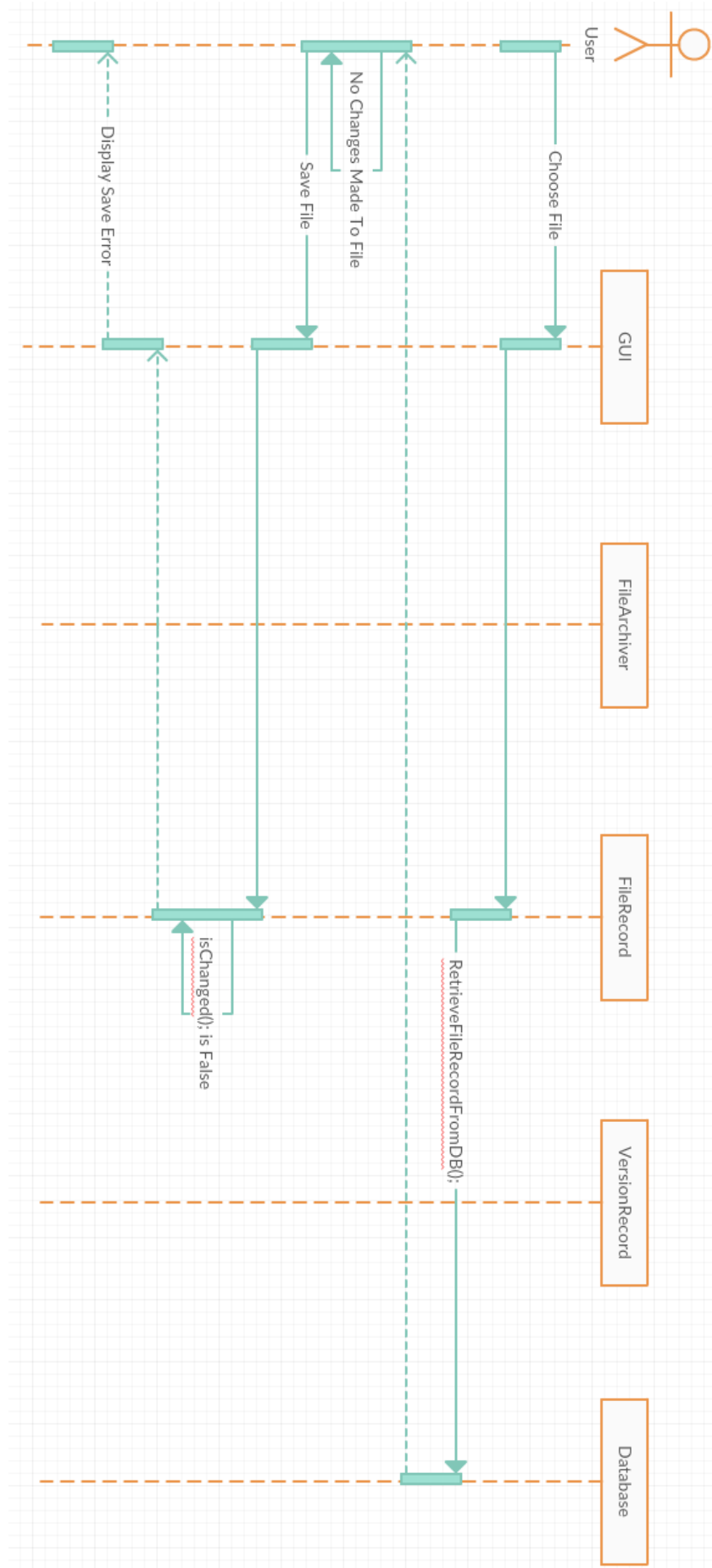
## UPDATING A CURRENT FILE IN THE SYSTEM.

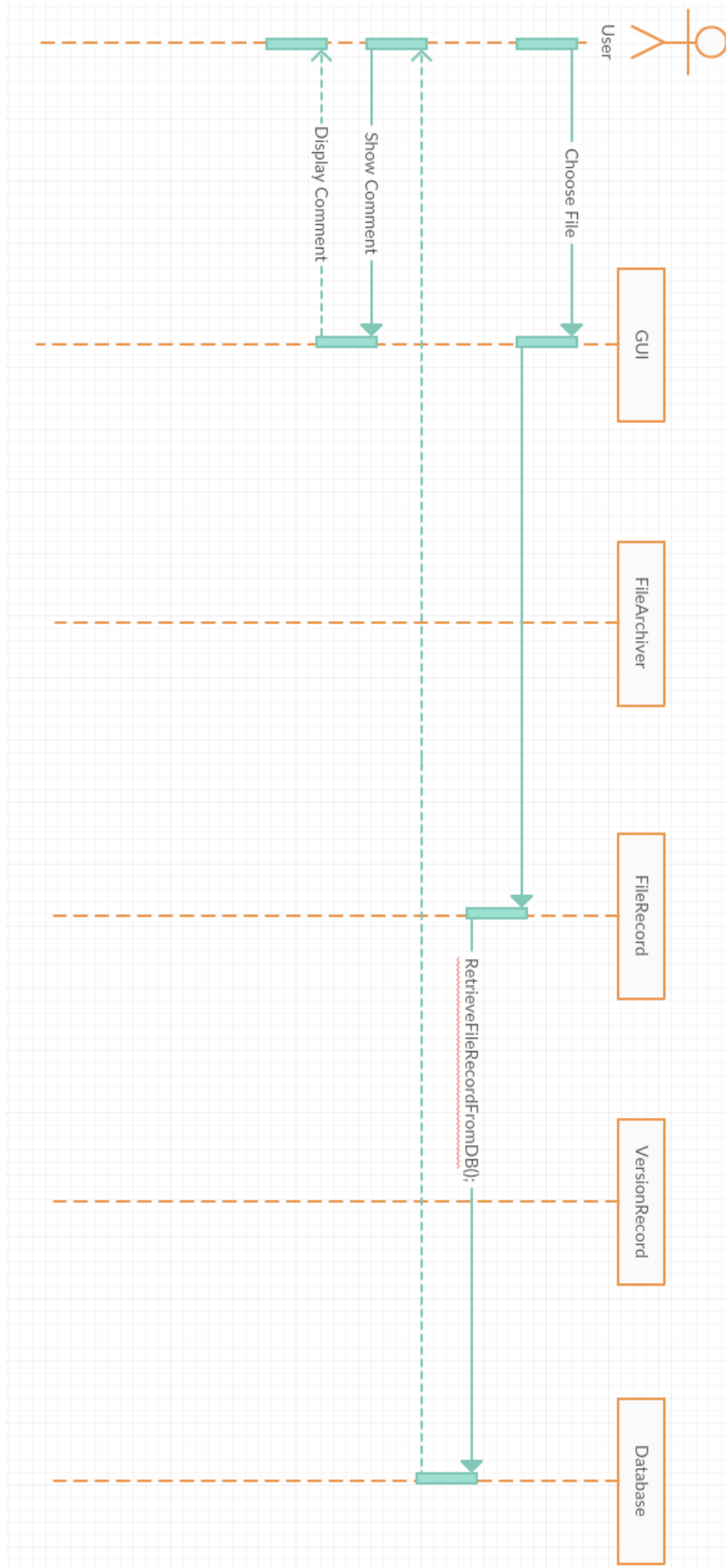


## RETRIEVING A CURRENT VERSION AND SETTING AS REFERENCE

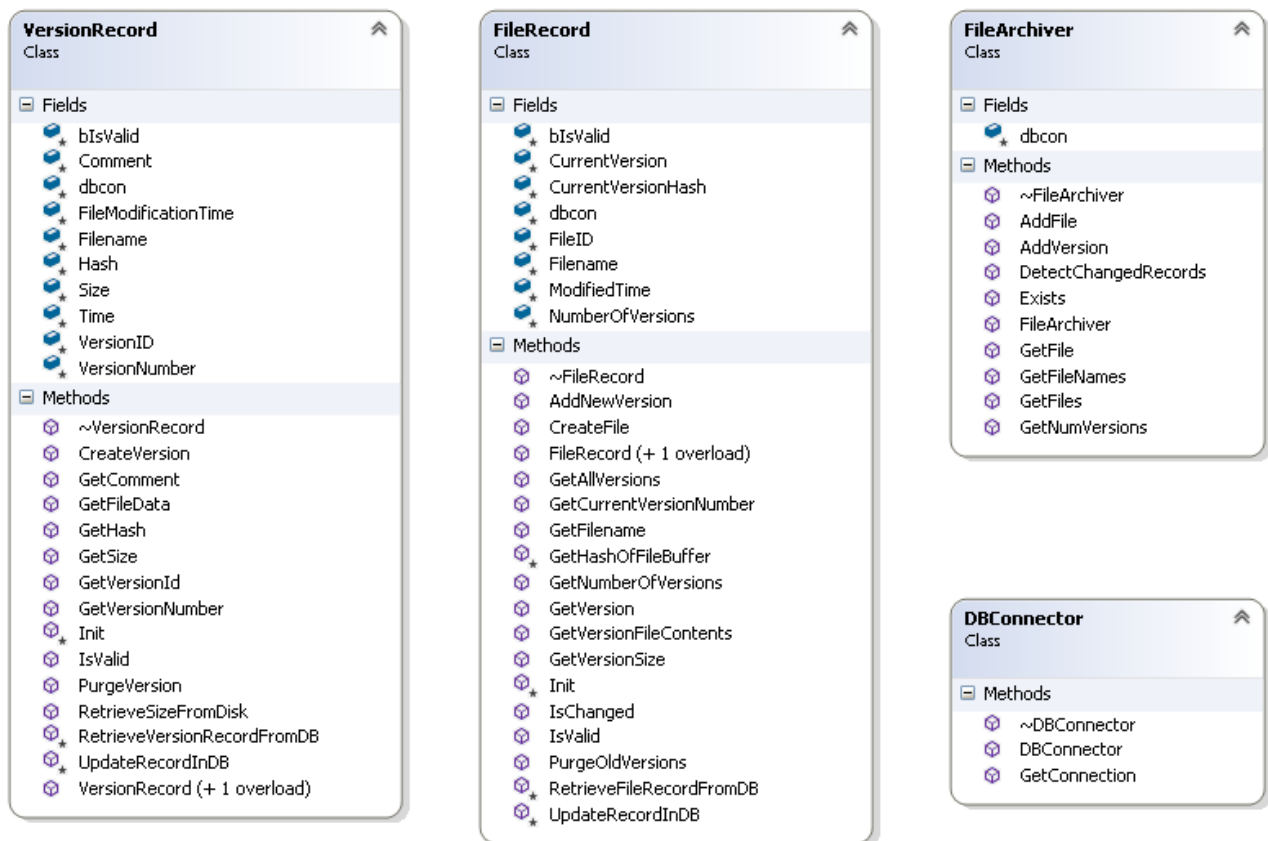


## ATTEMPTING TO SAVE WITHOUT HAVING MADE CHANGES

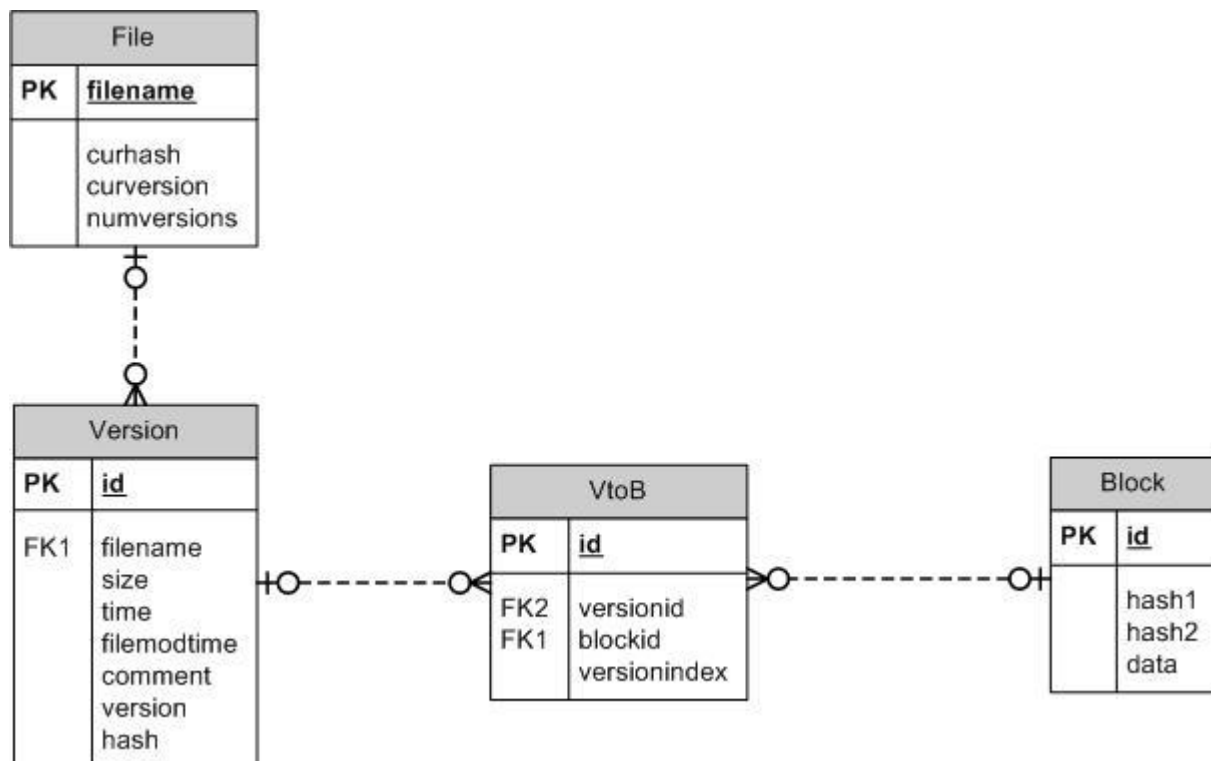




## BACKEND CLASS DIAGRAM



## DATABASE DESIGN – ENTITY RELATIONSHIP DIAGRAM



## DESIGN CHANGES

Our final database structure and specifics varied from the provided design and diagrams within the project documentation, the above image shows that final structure. Throughout the design process of the project we identified several key areas where we felt the design could be improved to allow the program functionality to be easier to implement and reduce possibilities of corruption of data. These changes all had the ultimate goal of simplifying the design and implementation process and to ensure we were creating a high quality, functional program by the end of the project.

## DESIGN ITERATION BREAKDOWN

### ITERATION 1

12/09/15 – 17/09/15:

Once the members met for the first time, a thorough overview of the program design was collaborated on a whiteboard, producing the UML diagrams. After breaking down the design, Thomas then created the database and populated it with dummy data, and performed the necessary checks to ensure the integrity of the database.

While the database was being configured for the project, each member was assigned to write the skeleton of the components previously identified in the overview:

- Ivana: created the stubs for the VersionRecord class.
- Josh: created the stubs for the FileRecord class and began to work on the FileLib class so that it was ready for use in a range of classes.
- Phil E: created the stubs for the FileArchiver class.

The members went home later this day and were assigned tasks to continue on with. Members collaborated over Skype, keeping up to date with one another.

- Thomas completed the code responsible for connecting the FileArchiver class to the database.
- Phil E completed the base functionality of the FileRec and FileArchiver classes. At this stage, the backend of the program was able to handle the creation of the blob data given a file and retrieve it back to some local file system, building off of FileRec, which stores the actual data that FileArchiver passes to/from the database.
- Ivana began to create the base GUI, while Josh was working on FileLib and CPP unit tests for it.
- On completion of FileLib and its tests, Josh and Ivana collaborated to complete the base GUI, which on completion used the functionality produced by Thomas and Phil. The GUI was tested completely functional in terms of displaying what was in the database.
- Phil M and Nicholas collaborated to create the diagrams that were outlined during the initial meeting (where the UML diagrams were drawn up as a group). The integrity of each diagram was verified as a group post-production.

## ITERATION 2

18/09/15 – 21/09/15:

During the implementation of the version control (VersionRecord and related components) in regards to the design of the code and the database, we came to the realisation that we can design it more concisely. So, the database design was refactored in the sense that redundant functionality was omitted.

- Thomas created an ERD of the new design. Josh completed FileLib at this point, but some of the functionality was omitted in source files relying on it in the end, as better implementations became apparent.
- Thomas and Phil E worked on refactoring the definitions of VersionRecord.
- Ivana and Josh worked on creating and connecting functionality that wraps the new logic of VersionRecord, which was completed.
- Nicholas worked on compiling notes and screenshots taken by the other members and himself.
- Phil M worked on creating the sequence diagrams of the new design.

21/09/15 – 27/09/15:

At this stage, most of the project has been completed. There are some important aspects of the version control to be implemented, but overall, the project is close to completion.

- Thomas created functionality responsible for file/data compression, which also includes the side-functionality of temporary storage being created during the compression/decompression phase of the interaction between the client and the server/database.
- Ivana and Josh completed the functional requirements of the GUI. All buttons slot into their related functions, and all probable user cases have been identified and handled thoroughly. Most dialogs notifying users of error have been completed.
- Phil E and Josh collaborated to write CPP unit tests for the functional requirements of the backend. Completed and tested thoroughly.

Josh noticed some logic errors associated with version purging, and notified Thomas to fix. It was fixed in a matter of minutes.

Phil E noticed a troublesome segmentation fault in a case where you close the window without doing anything when blackbox testing the GUI, so Josh and Ivana debugged to find a fault in the destructor of the MyWindow class that allowed deletion of a form that hadn't been opened. This was fixed promptly.

- Ivana applied final dialogs for user-error cases in the form of message dialogs. The GUI is now complete in all aspects, apart from one core functionality (old version purging). The stub and button have been interconnected as required, encapsulating pseudo-code of the functionality in VersionRecord responsible for the purging.
- Thomas finalized his part of the iteration by cleaning up code related to compression (CompressionUtils) and also fixed it to use gzip instead of zip.

- Josh, Phil E, and Ivana went through and did a final round of testing while also cleaning up useless comments and redundant code.
- The group was notified over Skype of the progress made: almost complete.
- Nicholas and Phil M are compiling what is required for the report, and begin writing it up.

### ITERATION 3

The project is practically complete: final touching up and functional testing to finish off, along with the addition of the final component for the project overall, which is also core functionality of the GUI. That is, `MyWindow::SetReferenceVersion` (use of `VersionRecs::PurgeOldVersions` in conjunction with Qt components) has to be completed. Finally, the report needs to be worked on.

27/09/15 – 29/09/15:

- Thomas identified an insertion issue with the database. This was fixed promptly. Thomas then finalised the functionality responsible for purging a given number of old version.
- Josh and Ivana wrapped the purging functionality that Thomas wrote up to work within the GUI and its components.
- Phil E requested that Josh adds functionality to `FileLib` (as he is responsible for the class) for use in the backend CPP unit tests and main source file of the program to setup and destroy temporary directories used for the compression/decompression phase when interacting with the database. Phil E wrote up the prototype to outline what it should do, and Josh implemented it into `FileLib` as needed.
- The group as a whole was notified over Skype of the completion of the design and code aspect of the project. The group has confirmed the integrity of the overall project, and the report can begin to be written.
- Nicholas and Phil M have working on the report during this time.
- All other members have now been allocated sub-tasks for the report: collaborating to finalise

### ELEMENT LIST

1. File record functionality – (Full functionality Completed during iteration 2)
  - a. Create file records – (Completed iteration 1)
  - b. Update file records – (Completed iteration 2)
  - c. Show version of files – (Completed iteration 2)
  - d. Remove versions – (Completed iteration 2)
  - e. Add new versions of files – (Completed iteration 2)
  - f. Show size information – (Completed iteration 1)
  - g. Check if a version has changed – (Completed iteration 1)
2. Version record functionality
  - a. Retrieve version record from database (Completed iteration 1)
  - b. Update Version information from a database (Completed iteration 2)
  - c. Create new version records (Completed iteration 1)
  - d. Show file size information (Completed iteration 1)



- e. Remove a version from the database (Completed iteration 2)
3. Gui Functionality (full functionality with data integration completed iteration 3)
  - a. Provide graphical interface for workings of the program (Gui base completed at iteration 1)
4. File Archiver (Full functionality achieved during iteration 3)
  - a. Check for already existing files (Completed during iteration 2)
  - b. Get version info (completed at iteration 2)
  - c. Insert a new file into database (Completed at iteration 2)
  - d. Add versions to files (completed at iteration 2)
  - e. Retrieve files and information from database (completed iteration 2 – at iteration 1-local file storage)
  - f. Compressing and uncompressing files from the database (Completed at iteration 2)

## UNIT TESTING PROCEDURES

Throughout the project various Unit testing functions were utilized to verify correct functionality of areas of the project.

A test for each bit of functionality was designed and utilized before the final commit to the repo and the item marked as completed on the internal TODO list maintained

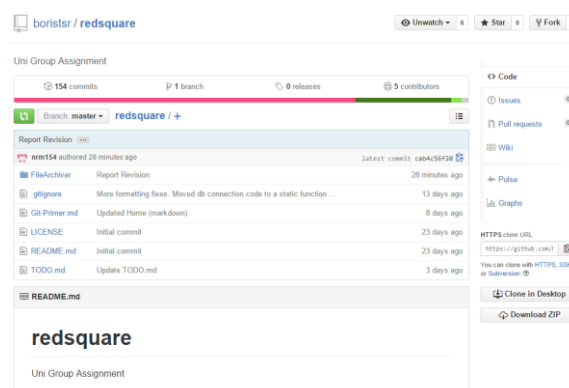
The additional testing functionality created by members was:

1. Testing of functionality when appending a path
2. Testing of functionality when querying a file name
3. Testing for proper hashing of files
4. Testing for accuracy of files modify dates
5. Testing for accuracy when querying the path of a file
6. Testing of successful file compression/decompression
7. Testing for successful commit and retrieve operations
8. Testing for successful removal operations
9. Testing for correct retrieval and accuracy of returned file information

## VERSION CONTROL

For version management we have decided to utilise GitHub, a web based Git repository hosting service. We have chosen GitHub due to most members of the group having particular familiarity with it and it's easy to use desktop application.

This is the main screen of the repository we used on the website.



## SETUP

GitHub makes setting up a local repository simple.

Firstly everybody in the group created a GitHub account, Phil E then created the repository 'redsquare' and added each group member to it for proper version control, after this the group is free to pull from the master branch and push/commit any new files or changes made to old ones.

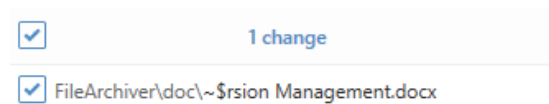
For Windows and Mac users there is a downloadable desktop application (Displayed above) for GitHub which makes this version management much easier.

## DETAILS/HOW WE USED IT

This black bar is the top level of the Git repository. It shows the entire timeline of commit history to the repository. Each dot represents a commit and clicking on each takes you back to the previous versions of the repository. This is great for data security to ensure nothing is lost in an accidental commit of wrong work.



The image below shows any uncommitted changes a user has made to a current document or file. To commit it to the repository they must include a summary of what they are adding/changing and a description so that other users know what changes have been made. We used this information to go over each other's work and make any recommendations to fix any mistakes the user who committed may have made and suggestions on how to fix them, working together to help each other.



Summary

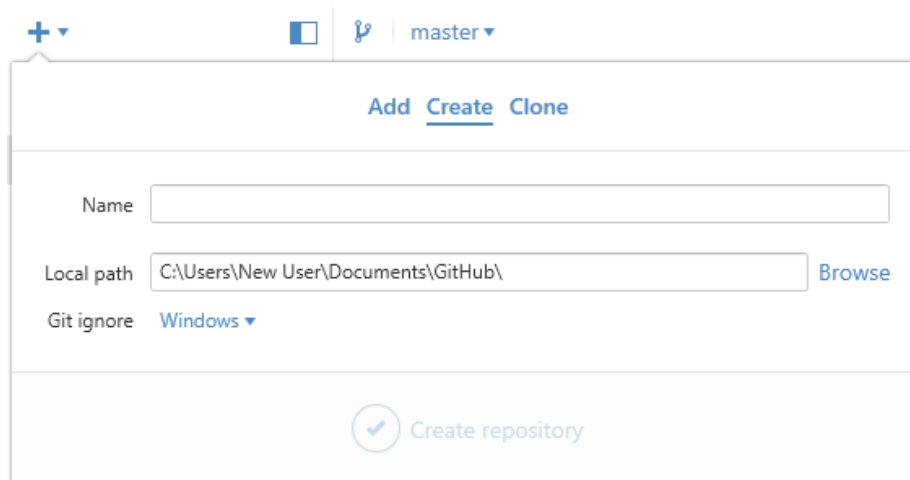
Description

☒ Commit to master

## ACCESS AND USAGE INFORMATION

GitHub makes accessing the repository easy whether it's through command line or through the desktop application. The website supplies a clone/checkout URL for HTTPS, SSH and Subversion which makes it easy for all different users of the repository.

The desktop application is a small 100mb download and makes the version management much easier. All the user has to do is click the plus '+' symbol on the top left of the GUI (Pictured below) and that will drop down a box whether the user wants to Add, Create or Clone a repository, then it's as simple as copying the clone URL into the program and then you're ready to go with all files from the repository.



For command line users there are various git commands that do everything needed, from pulling the files down as well as pushing them back up and committing changes. This Git Primer file was created by Phil E to help the other group members who are new to Git. It includes step by step instructions on what exactly we needed to do if we were on Linux or without the desktop application.

## Git Primer

If you want to use the command line the majority of commands you'll need are listed here.

### Clone the repository

```
git clone https://github.com/boristsr/redsquare.git
```

### Update your local copy with changes others have submitted

```
git pull
```

### To see locally changed, added or deleted files

```
git status
```

### Committing changes

When you want to commit changes you need to stage files for committing, run `git status` to check what files are changed. Then to add the files you want type `git add <files>` Wild characters work here `git add *.cpp *.h`

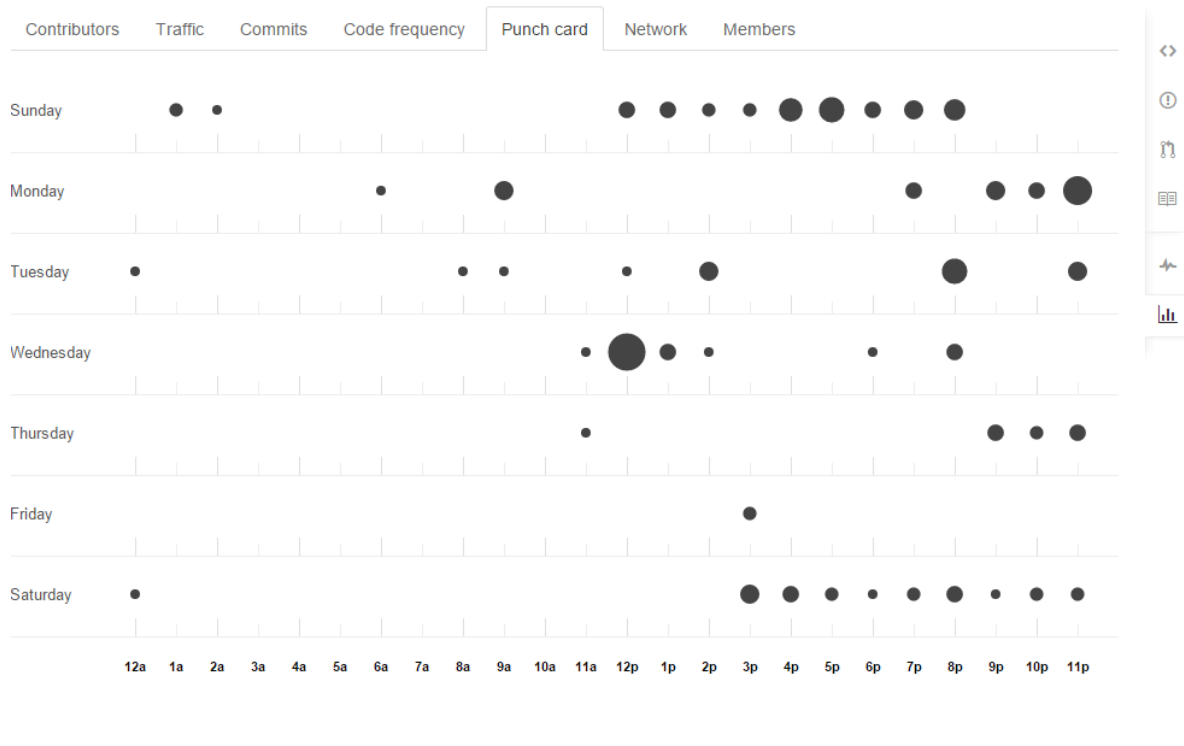
### Committing

Since git is a distributed VCS, you need to commit your changes to your repository, then push it to the main repository.

```
git commit -m "Put a comment here about what you've changed"
```

```
git push origin master
```

The image below is from the GitHub website. It shows the virtual punch card of work commitments the group has made. It shows which day and at what time each commit occurred at for our group it shows that most work is done in the afternoons and later parts of the day.

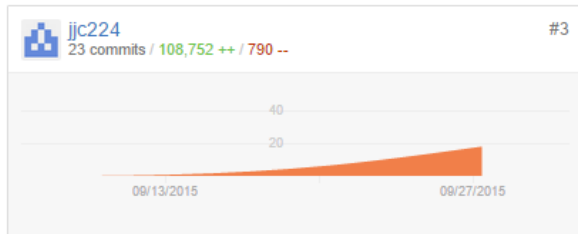
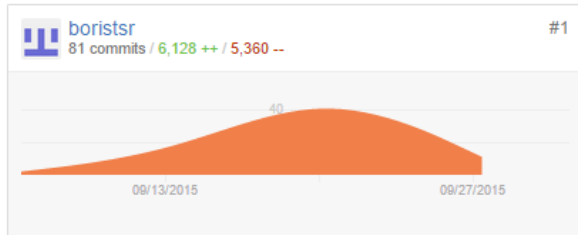
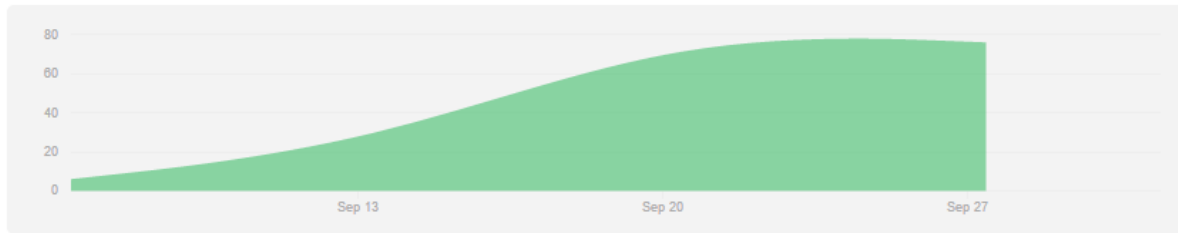


This is the final shot of the contribution statistics for all members, showing all members making successful use of GitHub versioning systems

Sep 6, 2015 – Oct 1, 2015

Contributions to master, excluding merge commits

Contributions: Commits ▾



**\*\*NOTE:** line count was broken by a mistake in committing a RTF file to the repo, line count for contribution for members nrm154 and jjc224 is inaccurate

## GROUP RECORDS

### GROUP MEETING SUMMARY

#### MEETING ONE

Meeting called by: Nicholas

Note taker: Nicholas

Timekeeper: Nicholas

Attendees: All

Report Presenter: Nicholas

Target Meeting Time: 30 minutes

#### **Manager's perception of project state:**

Being the first group meeting, my impressions mostly revolved around how I thought we would work as a group, some members had already worked together on projects, so group cohesion was already in a good place, the main adjustments required was introduction of the new people to the group.

**Agenda Topic:** Role selection

**Time Allotted:** 10 minutes

#### **Discussion:**

We took some time to discuss as a group which kind of role within RUP each member would like to take.

#### **Action Items:**

1. Advise of desired role within the group, by next week lab
2. Revise and familiarise yourself with the assignment details

#### **Agenda Review:**

Currently no items for review

#### **Agenda Topic:**

In preparation for the next week to come, the design of the project had to be outlined and understood by each individual within the group.

**Time Allotted:** 20 minutes

#### **Discussion:**

We took some time here to quickly review the assignment and began to brainstorm some ideas for the design.

#### **Action Items:**

1. Discuss and outline initial design thoughts for the project
2. Upload a concise coding style to enhance code consistency

#### **Agenda discussion:**

Some concerns were raised about the structure of the current design; a decision was made that we would revise the design in order to streamline it. No date was targeted, merely we acknowledge that we felt there was a need for redesign and agreed to go about it, at some undefined later date.

Philip Edwards also posted a coding style document for all members to follow; this was to ensure consistency in the code throughout the project.

---

## MEETING TWO

Meeting called by: Nicholas

Note taker: Nicholas

Timekeeper: Nicholas

Attendees: All

Report Presenter: Nicholas

Target Meeting Time: 30 minutes

### **Manager Perception of project state:**

At this stage my perceptions were largely the same as the previous meeting, with no hard work being done, there is no data to base a review on. The cohesion of the group appeared to be strong and members seemed flexible and willing to assist in all parts of the project.

**Agenda Topic:** Review the requirements of the project and handle role allocation.

**Time Allotted:** 15 minutes

### **Discussion:**

We took time to outline the requirements of the project in the sense of the overall functionality and the interconnectedness of sub-projects to build the project as a whole.

### **Action Items:**

1. Reviewed assignment specifications
2. Role preferences for the project, and the assigning of them

### **Agenda Review**

1. All members reported familiarising themselves with the assignment specs, questions were raised which lead to the group reading through the assignment spec's to get everyone on same page before beginning planning and implementation

1. Roles
  - a. Manager- Nicholas Ross Morgan
  - b. Lead Designer – Josh Coleman
  - c. Lead Implementer- Phil Edwards
  - d. Designers – Assigned to all
  - e. Systems Integration – Ivana Ozakovic, Phil Edwards

\*Roles were expanded- check detailed report for final roles

**Agenda Topic:** Identify appropriate consultation times and collaboration methods

**Time Allotted:** 15 minutes

**Discussion:**

The group discussed ways to communicate, as well as methods to get in contact with one another if any of the services relied on fail.

**Action Items:**

1. Assign regular meeting times
2. Establish secondary communication methods
3. Investigate Technologies for use( MySQL/Mongo, GitHub/provided repo)

**Agenda discussion**

1. Regular meeting times were scheduled for 6 PM every Sunday, with regular informal check-ins during the week
2. Group chose Skype as a means for communication outside of physical meetings
3. Phone numbers were exchanged as a backup and for on-the-go contact
4. A brief discussion was had about the familiarities of each member with the systems, members reported being more familiar with MySQL and GitHub came out as the preferred version management

---

**MEETING 3**

Meeting called by: Thomas Nixon

Note taker: Phil

Timekeeper: Thomas

Attendees: Phillip Edwards, Thomas Nixon, Joshua Coleman, and Ivana Ozakovic

Report Presenter: Nicholas Morgan

Target Meeting Time: 1 Hour

**Manager Perception of project state:**

I was not present for this meeting, but after receiving the notes and seeing the outcome I was happy with the direction the project was taking, a complete redesign on the structure had been completed.

**Agenda Topic:**

Redesign of the overall system.

**Time Allotted:** 30 Minutes

**Discussion:**

Integrity insurance of systems.

**Action Items:**

1. Supporting system selection
2. Ensure all members have installed Git to manage/share work on the project

**Agenda Review**

1. Decisions were made based on the preferences of the group at the previous meeting to go ahead with usage of MySQL and GitHub



2. Integrity of the server/database was confirmed accessible by all members individually, then reported the success to one another as a group.
3. A Git primer was added to the project's GitHub to assist members in their usage

**Agenda Topic:** Overall project structure.

**Time Allotted:** 20 minutes.

**Action Items:**

1. Project structure redesign
2. Database setup

**Agenda discussion**

1. A lengthy discussion was had about the redesign with a complete redesign of the structure complete by the end of the meeting \*refer to detailed report section
2. A database was setup to allow testing of interactions with code and a live database

---

**MEETING 4**

Meeting called by: Nicholas

Note taker: Nicholas

Timekeeper: Nicholas

Attendees: All

Report Presenter: Nicholas

Target Meeting Time: 1 Hour

**Manager Perception of project state:**

At this stage prototyping for functions within multiple sections of the program had been completed, the design phase was completed and members had a clear view of the new direction of the assignment, The project as a whole seemed to be on track to be completed. Group cohesion was slightly lower, with the redesign happening in the previous meeting, however by the end of the week's meeting, people were back on the same page.

**Agenda Topic:** A continuation of the design to ensure integrity of interconnected components.

**Time Allotted:** 35 minutes

**Discussion:**

More thorough outline of the overall design.

**Action Items:**

1. Discussion on the changes made to design
2. GitHub member familiarization

**Agenda Review**

1. Phil and Thomas ran the members not present at the previous meeting through the specifics of the redesign

2. Time was spent showing members the proper use of Git to avoid issues and insure all could successfully commit their work without issue

**Agenda Topic:** Project code and the initial assignment of it.

**Time Allotted:** 20 minutes

**Discussion:**

Discussion of the actual code and who should be assigned the relative sub-tasks.

**Action Items:**

1. Initial Code Assignment and discussion

**Agenda discussion:**

1. Code Assignments
  - a. VersionRecord – Thomas
  - b. GUI/Reporting – Nicholas
  - c. FileRecord – Phillip E.
  - d. CPP Unit testing– Ivana/Josh
  - e. Gui - Phillip Mihajlovski.

---

## MEETING 5

Meeting called by: Nicholas

Note taker: Nicholas

Timekeeper: Nicholas

Attendees: All

Report Presenter: Nicholas

Target Meeting Time: 1 Hour

**Manager Perception of project state:**

Project is currently in crunch mode, some deliverables of members hadn't been committed on time, other group members had been working hard and a large portion of coding was completed, at the beginning of this meeting CPP Unit testing, GUI and some touch ups on the VersionRecord and FileRecord functionality needed to be completed, reports also needed to be completed and the design document fleshed out.

**Agenda Topic:** Identification of possible issues with the functionality of the project thus far.

**Time Allotted:** 20 minutes

**Discussion:**

Identify issues in the design/functionality and handle as necessary, as well as identify what else follows.

**Action Items:**

1. Update internal TODO list
2. Discuss issues in functionality

**Agenda Review:**

1. As a group we sat down and updated the TODO list, this updated all members perspective of the current state and gave us a clear idea of what was remaining to finish
2. We took some time as a group to discuss any functionality issues we had in an effort to see if any group member could provide insight or a new look on the problem

**Agenda Topic:** An update of what components need to still be completed.

**Time Allotted:** 25 Minutes

**Discussion:**

An overall update of the project: what components have been completed.

**Action Items:**

1. GUI design and implementation finalized
2. CppUnit testing code finalization
3. Makefile fixes
4. FileRec and VersionRecord implementation finalized
5. Reports
6. Presentation

**Agenda discussion:**

1. GUI design was passed onto Ivana and Josh
2. Cpp Unit testing was assigned to all members
3. Makefile fixes for Unit testing
4. Report writing assigned to Nicholas
5. FileRecord and VersionRecord finalized assigned to Thomas and Philip Edwards
6. Presentation for inclusion in report assign to Ivana
7. Final meeting

**Meeting called by:** Nicholas

**Note taker:** Nicholas

**Timekeeper:** Nicholas

**Attendees:** All

**Report Presenter:** Nicholas

**Manager Perception of project state:**

Project is currently in crunch mode, some deliverables of members hadn't been committed on time, other group members had been working hard and a large portion of coding was completed, at the beginning of this meeting Cpp Unit testing, GUI and some touch ups on the versionrec and filerec functionality needed to be completed, reports also needed to be completed and the design document fleshed out.

**Agenda Topic:** Final check to ensure all tasks have been completed.

**Time Allotted:** 20 minutes

**Discussion:**

Verify with one another what has been completed and check that no incomplete tasks remain.

**Action Items:**

1. Final check of TODO list
2. Report update with finalized info

**Agenda Review:**

1. As a group we went through the TODO, ensure all items checked off were completed and assigned final tasks
2. Tasks
  - a. GUI – Joshua and Ivana
  - b. Report – Phil Mihajlovski and Nicholas
  - c. VersionRecord – Philip Edwards
  - d. FileRecord – Thomas
  - e. CPP Unit testing – Phil Edwards, Ivana Ozakovic, and Joshua Coleman

## GROUP MEMBER WORK JOURNAL SAMPLES

Group members maintained a work journal to track work times, the primary goal of these journals was to identify and shortcoming pertaining to time management and to ensure an organized approach to the design and implementation of the project.

### SAMPLE OF WORK DIARY FROM THOMAS NIXON

Work Diary

Author: Thomas Nixon

6/09/2015

Read documentation

Created Google doc to start putting outlines of what needs to be done in.

Researched MySQL and its use with c++

Time: 3:00pm-7:00pm

Created FileArchive.h & FileArchive.cpp.

Defined class members and modified names.

Insured code was correct.

Time: 7:00pm-7:30pm

Created a rough version of the database MySQL code.

Currently untested and needs to be checked.

Time: 7:30pm-8:00pm

14/09/2015

Put SQL in MySQL server.

Fixed a few small errors in the code.

Tested operation.

Database ready to be used.

Time: 9:00pm-10:00pm

15/09/2015

Wrote the code to connect to MySQL server.

Tested operation over internet and returned data.

Updated some of the class in the FileArchiver with the code.

The functions are not complete but have the frame work to used.

Time: 9:00pm-11:00pm

## Work Diary

**Author:** Ivana Ozakovic

---

### Work#:1

**Short description:** Declare functions and data members for VersionRecord.h and set up function stubs in VersionRecord.cpp

#### Planned work schedule

Date: 12/09/15, Saturday

Time: Anytime

#### Actual work time

Date: 12/09/15

Time: 12:30pm - 1:30pm

#### Summary of work completed

**Part of the Project:** Implementation

**Files modified:** VersionRecord.cpp (created), VersionRecord.h

**Functions modified:** /

**Description:** Declared data members and functions in the VersionRecord.h.

Created VersionRecord.cpp file and set up stubs for declared functions.

**Defects/problems:** Functions have to be defined.

---

## SUPPORTING CODE SAMPLES

### SUPPORT CODE FOR CONNECTING TO THE DATABASE

```
// Connect to database
try
{
    driver = get_driver_instance();
    dbcon = driver->connect(host, user, pw);
    dbcon->setSchema(dbname);
    bInitialised = true;
}
catch (sql::SQLException& e)
{
    log("ERROR: ");
    log(e.what());
    log(boost::lexical_cast<string>(e.getErrorCode()));
    log(e.getSQLState());
}

//dbcon->setTransactionIsolation(sql::TRANSACTION_READ_UNCOMMITTED);
return dbcon;
```

### SUPPORT FOR UTILITIES FOR LOGGING DEBUG INFO

```
#include "ProjectConstants.h"

void logToFile(std::string message)
{
#ifdef DEBUG_LOG_TO_FILE
    std::ofstream outFile("debuglog.log", std::ios::app);
    if(outFile.is_open())
    {
        outFile << message << std::endl;
        outFile.close();
    }
#endif // DEBUG_LOG_TO_FILE
}

void log(std::string message)
{
#ifdef DEBUG_LOGGING
    std::cout << message << std::endl;
    logToFile(message);
#endif //DEBUG_LOGGING
}
```

## Support code sample - hasing - Murmurhash

```
void MurmurHash3_x86_32 ( const void * key, int len,
                          uint32_t seed, void * out )
{
    const uint8_t * data = (const uint8_t*)key;
    const int nblocks = len / 4;

    uint32_t h1 = seed;

    const uint32_t c1 = 0xcc9e2d51;
    const uint32_t c2 = 0x1b873593;

    //-----
    // body

    const uint32_t * blocks = (const uint32_t*)(data + nblocks*4);

    for(int i = -nblocks; i; i++)
    {
        uint32_t k1 = getblock32(blocks,i);

        k1 *= c1;
        k1 = ROTL32(k1,15);
        k1 *= c2;

        h1 ^= k1;
        h1 = ROTL32(h1,13);
        h1 = h1*5+0xe6546b64;
    }

    //-----
    // tail

    const uint8_t * tail = (const uint8_t*)(data + nblocks*4);

    uint32_t k1 = 0;

    switch(len & 3)
    {
        case 3: k1 ^= tail[2] << 16;
        case 2: k1 ^= tail[1] << 8;
        case 1: k1 ^= tail[0];

        k1 *= c1; k1 = ROTL32(k1,15); k1 *= c2; h1 ^= k1;
    };

    //-----
    // finalization

    h1 ^= len;

    h1 = fmix32(h1);

    *(uint32_t*)out = h1;
}
```



## UNIT TESTING SAMPLES

### TESTING FOR CORRECTLY APPENDED PATH TO FILE

```
void newtestclass::testAppendPath()
{
    const std::size_t NUM_PATH_ELEMENTS_1 = 2;
    const std::size_t NUM_PATH_ELEMENTS_2 = 3;

    std::string pathElements1[NUM_PATH_ELEMENTS_1] = {"/test", "path1"};
    std::string pathElements2[NUM_PATH_ELEMENTS_2] = {"another", "path2", "test"};

    std::string path1          = generateUnnormalizedPath(pathElements1, NUM_PATH_ELEMENTS_1);
    std::string path2          = generateUnnormalizedPath(pathElements2, NUM_PATH_ELEMENTS_2);
    std::string expectedResult = "/test/path1/another/path2/test/";
    FileLib fileLib;
    std::string result = fileLib.AppendPath(path1, path2);

    std::cerr << "result = " << result << std::endl;
    CPPUNIT_ASSERT(result == expectedResult);
}
```

### TESTING FOR RETRIEVING FILENAME FUNCTIONALITY

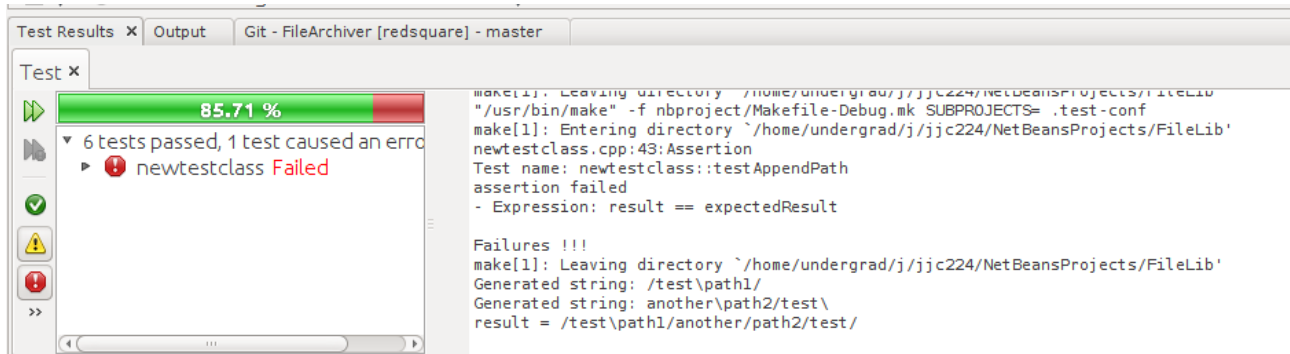
```
void newtestclass::testGetFilename()
{
    const std::size_t NUM_PATH_ELEMENTS = 2;
    std::string pathElements[NUM_PATH_ELEMENTS] = {"/some", "path"};

    std::string file = "file.ext";
    std::string path = generateUnnormalizedPath(pathElements, NUM_PATH_ELEMENTS) + file;
    FileLib fileLib;
    std::string result = fileLib.GetFilename(path);
    std::string expectedResult = file;

    CPPUNIT_ASSERT(result == expectedResult);
}
```

## BUG/TESTING LOG SAMPLES

Sample from a run of some implemented unit testing code



We took an iterative approach to our design, starting from basics and improving upon them, during our design we encountered bugs, the two diagrams below show the occurrence of a bug (improper extensions for a boost function) the first resulted in some unexpected behaviour, with the second image showing the solution after the bug fix

```
// Replaces one or more backslashes and two or more forward slashes with a single forward slash.
string FileLib::Normalize(string path)
{
    boost::regex re("\\\\+|//+");
    path = boost::regex_replace(path, re, "/");

    return path;
}
```

```
// Replaces one or more backslashes and two or more forward slashes with a single forward slash.
string FileLib::Normalize(string path)
{
    boost::regex re("\\\\+|//+");
    path = boost::regex_replace(path, re, "/", boost::match_default | boost::format_all);

    return path;
}
```

## APPENDICES

### CODE ELEMENTS

FileArchiver.cpp / FileArchiver.h

MyWindows.cpp / MyWindow.h / MyWindow.ui

FileLib.cpp / FileLib.h

ProjectConstants.h

FileRecord.cpp / FileRecord.h

RetrieveForm.cpp / RetrieveForm.h

CompressUtils.cpp / CompressUtils.h

TestUtilities.cpp / TestUtilities.h

DBConfigurationFileUtility.cpp

Utilities.cpp

DBConnector.cpp / DBConnector.h

VersionRecord.cpp / VersionRecord.h

GetCommentForm.cpp / GetCommentForm.h

FileLibTester.cpp / FileLibTester.h

MurmurHash.cpp / MurmurHash.h

BackendTests.cpp / BackendTests.h



```

/*
 * File:    MyWindow.cpp
 * Author:  io447
 *
 * Created on 21 September 2015, 12:36 PM
 */
#include <QFileDialog>
#include <QApplication>
#include <QMessageBox>
#include <iostream>
#include <string>
#include <QDebug>
#include <boost/lexical_cast.hpp>
#include <QStandardItemModel>

#include "MyWindow.h"
#include "FileArchiver.h"
#include "FileRecord.h"
#include "FileLib.h"

MyWindow::MyWindow() {
    widget.setupUi(this);

    widget.saveCurrentBtn->setEnabled(false);
    widget.retrieveVersionBtn->setEnabled(false);
    widget.setReferenceBtn->setEnabled(false);
    widget.showCommentBtn->setEnabled(false);

    //connect SelectFile() with selectFileBtn
    connect(widget.selectFileBtn, SIGNAL(clicked()), this, SLOT(SelectFile()));
//connect saveCurrentBtn with SaveCurrent()
connect(widget.saveCurrentBtn, SIGNAL(clicked()), this, SLOT(SaveCurrent()));
//connect RetrieveVersion() with retrieveVersionBtn which will open RetrieveForm
    connect(widget.retrieveVersionBtn, SIGNAL(clicked()), this, SLOT(RetrieveVersion()));

//connect setReferenceBtn with SetReferenceVersion() to delete unnecessary file versions
connect(widget.setReferenceBtn, SIGNAL(clicked()), this, SLOT(SetReferenceVersion()));
//connect ShowComment() with showCommentBtn
connect(widget.showCommentBtn, SIGNAL(clicked()), this, SLOT(ShowComment()));
}

MyWindow::~MyWindow() {
}

//file selection
void MyWindow::SelectFile() {
    //declare new select file dialog
    QFileDialog dialog(this);
    //set mode to existing file
    dialog.setFileMode(QFileDialog::ExistingFile);
    //set view mode to detail
    dialog.setViewMode(QFileDialog::Detail);

```

```

QStringList fileNamees;

if (dialog.exec() == QDialog::Rejected)
{
    return;
}

fileNamees = dialog.selectedFiles();

//Display name of file as chosen by user
if(!fileNamees.isEmpty())
{
    fileName = fileNamees[0];
    widget.selectFilePath->setText(fileName);
}

//convert to std string
std::string stdFileName;
stdFileName = fileName.toStdString();

//for now not catching exception bad_alloc
FilePtr currentPath = new FileArchiver;

//If a record already exists
if(!currentPath->Exists(stdFileName))
{
    CreateFirstVersion(stdFileName);
}

RetrieveVersionDataForFile();
}

void MyWindow::SaveCurrent()
{
    FileRecord fileRec(fileName.toStdString());

    if(fileRec.IsChanged())
    {
        AddNewVersion(fileName.toStdString());
        RetrieveVersionDataForFile();
    }
    else
    {
        QMessageBox msgBox;

        msgBox.setWindowTitle("No save required.");
        msgBox.setText("No modifications since last version. No save necessary.");
        msgBox.setStandardButtons(QMessageBox::Ok);
        msgBox.exec();
    }
}

void MyWindow::ShowComment()

```

```

{
    QModelIndexList indexes = widget.tableView->selectionModel()->selectedRows();

    if(!(indexes.size() > 0))
    {
        QMessageBox msgBox(QMessageBox::Information, "Error",
            "Please select the file version", QMessageBox::Ok, 0);

        msgBox.exec();
        return;
    }

    if(indexes.size() > 0)
    {
        //get version number of selected record
        QVariant data = indexes[0].data(0);
        //retrieve version

        VersionRecord selectedVersion(fileName.toStdString(), data.toInt());
        if(selectedVersion.IsValid())
        {
            QMessageBox msgBox(QMessageBox::Information, "Comment for selected version",
                QString(selectedVersion.GetComment().c_str()), QMessageBox::Ok, 0);

            msgBox.exec();
        }
    }
}

void MyWindow::SetReferenceVersion()
{
    QModelIndexList indexes = widget.tableView->selectionModel()->selectedRows();

    if(!(indexes.size() > 0))
    {
        QMessageBox msgBox(QMessageBox::Information, "Error",
            "Please select the file version", QMessageBox::Ok, 0);

        msgBox.exec();
        return;
    }

    if(indexes.size() > 0)
    {
        QMessageBox msgBox(QMessageBox::Question, "Set reference version",
            "Are you sure you want to purge these records?", QMessageBox::Yes |
            QMessageBox::No, 0);
        msgBox.setDefaultButton(QMessageBox::No);
    }
}

```

```

        if(msgBox.exec() == QMessageBox::Yes)
        {
            FileRecord fileRec(fileName.toStdString());
            const QAbstractItemModel *myModel = indexes[0].model();

            for(int i = myModel->data(myModel->index(0, 0)).toInt(); i <
myModel->data(myModel->index(indexes[0].row(), 0)).toInt(); ++i)
            {
                fileRec.GetVersion(i).PurgeVersion();
            }

            // Repopulate the table.
            RetrieveVersionDataForFile();
        }
    }
}

```

```
void MyWindow::CreateFirstVersion(std::string fileName)
```

```

{
    getCommentWindow = new GetCommentForm();
    QString comm;

    if(getCommentWindow->exec() == QDialog::Rejected)
    {
        return;
    }

    comm = getCommentWindow->GetComment();

    std::string commentStd = comm.toStdString();
    FileRecord fileRec;

    fileRec.CreateFile(fileName, commentStd);
}

```

```
void MyWindow::AddNewVersion(std::string fileName)
```

```

{
    getCommentWindow = new GetCommentForm();
    QString comm;

    if(getCommentWindow->exec() == QDialog::Rejected)
    {
        return;
    }

    comm = getCommentWindow->GetComment();

    std::string commentStd = comm.toStdString();
    FileRecord fileRec(fileName);

    comm = getCommentWindow->GetComment();
    fileRec.AddNewVersion(fileName, commentStd);
}

```



```

}

void MyWindow::RetrieveVersionDataForFile()
{
    FileRecord fileRec(fileName.toStdString());

    if(fileRec.GetNumberOfVersions() == 0)
    {
        return;
    }

    QStandardItemModel *myModel = new QStandardItemModel(fileRec.GetNumberOfVersions(), 3,
this);
    myModel->clear();

    myModel->setHorizontalHeaderItem(0, new QStandardItem(QString("Version #")));
    myModel->setHorizontalHeaderItem(1, new QStandardItem(QString("Date")));
    myModel->setHorizontalHeaderItem(2, new QStandardItem(QString("Size")));

    vector<VersionRecord> versionRecs = fileRec.GetAllVersions();
    unsigned int currentRow = 0;

    for(vector<VersionRecord>::iterator it = versionRecs.begin(); it != versionRecs.end(); ++it)
    {
        myModel->setItem(currentRow, 0, new
QStandardItem(QString(boost::lexical_cast<string>(it->GetVersionNumber()).c_str())));
        myModel->setItem(currentRow, 1, new
QStandardItem(QString(FileLib::GetFormattedModificationDate(fileRec.GetFilename()).c_str())));
        myModel->setItem(currentRow, 2, new
QStandardItem(QString(boost::lexical_cast<string>(it->GetSize()).c_str())));

        ++currentRow;
    }

    widget.tableView->setModel(myModel);
    widget.tableView->resizeColumnsToContents();
    widget.tableView->setEditTriggers(QAbstractItemView::NoEditTriggers);
    widget.tableView->setSelectionBehavior(QAbstractItemView::SelectRows);
    widget.tableView->setSelectionMode(QAbstractItemView::SingleSelection);

widget.saveCurrentBttn->setEnabled(true);
    widget.retrieveVersionBttn->setEnabled(true);
    widget.showCommentBttn->setEnabled(true);
    widget.setReferenceBttn->setEnabled(true);

    widget.tableView->show();
}

void MyWindow::RetrieveVersion()
{
    QString directory;
    QString outFilename;

```

```

QModelIndexList indexes = widget.tableView->selectionModel()->selectedRows();

if(!(indexes.size() > 0))
{
    QMessageBox msgBox(QMessageBox::Information, "Error",
        "Please select the version to be retrieved", QMessageBox::Ok, 0);

    msgBox.exec();
    return;

}

retrieveWindow = new RetrieveForm;
//execute RetrieveForm and details of where retrieved file will be placed

if(retrieveWindow->exec() == QDialog::Accepted)
{
    directory = retrieveWindow->GetDirectory();
    outFilename = retrieveWindow->GetOutputFilename();
}

//convert data from RetrieveForm to full file output path
std::string fullOutputPath;

fullOutputPath += directory.toStdString();
fullOutputPath += "/";
fullOutputPath += outFilename.toStdString();

//retrieve version
if(indexes.size() > 0)
{
    VersionRecord selectedVersion(fileName.toStdString(), indexes[0].row() + 1);
    selectedVersion.GetFileData(fullOutputPath);
}
}

```

```
/*
 * File:    ProjectConstants.h
 * Author: philipedwards
 *
 * Created on 19 September 2015, 11:01 PM
 */

#ifndef PROJECTCONSTANTS_H
#define PROJECTCONSTANTS_H

#include <string>
#include <fstream>
#include <iostream>

#define DEBUG_LOGGING
#define DEBUG_LOG_TO_FILE

#define NIXON_SNAKE

const int MURMUR_SEED_1 = 23455;
const int MURMUR_SEED_2 = 2086235969;
const int FILENAME_LENGTH(767);

const int BLOCK_SIZE = 4000;

void logToFile(std::string message);
void log(std::string message);

const std::string COMPRESSION_WORK_PATH = "./temp/";

#endif/* PROJECTCONSTANTS_H */
```

```
/*
 * File:    RetrieveForm.h
 * Author:  io447
 *
 * Created on 21 September 2015, 4:45 PM
 */

#ifndef _RETRIEVEFORM_H
#define _RETRIEVEFORM_H

#include "ui_RetrieveForm.h"

class RetrieveForm : public QDialog {
    Q_OBJECT
public:
    RetrieveForm();
    virtual ~RetrieveForm();
public slots:
    void SelectDirectory();
    void SetFileName();
    QString GetDirectory();
    QString GetOutputFilename();
private:
    Ui::RetrieveForm widget;
    QString directoryPath;
    QString outputFilename;
};

#endif/* _RETRIEVEFORM_H */
```

```
/*
 * File:    RetrieveForm.cpp
 * Author:  io447
 *
 * Created on 21 September 2015, 4:45 PM
 */

#include <QFileDialog>
#include <QApplication>
#include <iostream>
#include <string>

#include "RetrieveForm.h"

RetrieveForm::RetrieveForm() {
    widget.setupUi(this);

//select directory button
    connect(widget.pushButtonDirectory, SIGNAL(clicked()), this, SLOT(SelectDirectory()));

//ok | cancel buttons
    connect(widget.buttonBox, SIGNAL(accepted()), this, SLOT(SetFileName()));

    widget.buttonBox = new QDialogButtonBox(QDialogButtonBox::Ok
                                             | QDialogButtonBox::Cancel);

    connect(widget.buttonBox, SIGNAL(accepted()), this, SLOT(accept()));

    connect(widget.buttonBox, SIGNAL(rejected()), this, SLOT(reject()));
}

RetrieveForm::~~RetrieveForm() {
}

void RetrieveForm::SelectDirectory()
{
    //declare new select file dialog
    QFileDialog dialogDir(this);
    //set mode to existing file
    dialogDir.setFileMode(QFileDialog::Directory);
    //set view mode to detail
    dialogDir.setViewMode(QFileDialog::Detail);

    QStringList dirNames;
    //QString fileName;
    if (dialogDir.exec())
    {
        //dialog.selectFile(fileName);
        dirNames = dialogDir.selectedFiles();
    }

    //Display name of directory as chosen by user
    QString dirName;
```

```
        if(!dirNames.isEmpty())
        {
            dirName = dirNames[0];
            widget.lineEditDirectory->setText(dirName);
        }

directoryPath = dirName;

}

void RetrieveForm::SetFileName() {

    outputFilename = widget.lineEditFilename->text();

}

QString RetrieveForm::GetDirectory()
{
return directoryPath;
}

QString RetrieveForm::GetOutputFilename()
{
return outputFilename;
}
```

```
/*
 * File:    TestUtilities.h
 * Author:  philipedwards
 *
 * Created on 21 September 2015, 8:41 AM
 */

#ifndef TESTUTILITIES_H
#define TESTUTILITIES_H

#include <string>
#include <mysql_connection.h>
#include <mysql_driver.h>

void createFile(unsigned int seed, std::string filename, int length);
void appendFile(int seed, std::string filename, int length);

void DropTables();
void CreateTables();

void ExecuteSQLFile(std::string path);

bool ExecuteUpdateStatement(sql::Connection* dbcon, std::string sqlstatement);

void CommitFileWithOneVersion(std::string path);
void CommitFileWithTwoVersions();

bool GenerateFilesAndCommitVersionsAndVerifyRetrieval(std::string path, unsigned int size,
unsigned int numVersions);

void RunTestCommitFileOneVersion();
void RunTestCommitFileOneVersionRetrieve();
void RunTestCommitFileWithTwoVersionsRetrieveBoth();

void RunTestPurge();

#endif/* TESTUTILITIES_H */
```

```
#include <string>
#include "DBConnector.h"
#include "TestUtilities.h"
#include "ProjectConstants.h"
#include <fstream>
#include "FileRecord.h"
#include "MurmurHash3.h"

#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>
#include "boost/lexical_cast.hpp"

#include <cstdlib>

using namespace std;

void createFile(unsigned int seed, string filename, int length)
{
    srand(seed);

    ofstream outFile(filename.c_str());

    if(outFile.is_open())
    {
        while(length > 0)
        {
            length--;
            char num = rand() % 256;
            outFile.put(num);
        }
        outFile.close();
    }

    void appendFile(int seed, string filename, int length)
    {
        srand(seed);

        ofstream outFile(filename.c_str(), ios::app);

        if(outFile.is_open())
        {
            while(length > 0)
            {
                length--;
                char num = rand() % 256;
                outFile.put(num);
            }
        }
    }
}
```



```
void DropTables()
{
    ExecuteSQLFile("sql/DropTables.sql");
}

void CreateTables()
{
    ExecuteSQLFile("sql/CreateTables.sql");
}

void ExecuteSQLFile(string path)
{
    sql::Connection* dbcon = DBConnector::GetConnection();
    dbcon->setSchema("redsquare");

    if(dbcon == NULL)
    {
        log("Failed to open database connection");
        return;
    }

    ifstream inFile(path.c_str());
    if(!inFile.is_open())
    {
        log("Failed to open sql file");
        delete dbcon;
        dbcon = NULL;
        return;
    }

    while(!inFile.eof())
    {
        string statementstring;
        getline(inFile, statementstring);
        if(statementstring.empty() == false)
        {
            log(statementstring);
            ExecuteUpdateStatement(dbcon, statementstring);
        }
    }

    //inFile.close();
    //delete dbcon;
    dbcon = NULL;
}

bool ExecuteUpdateStatement(sql::Connection* dbcon, std::string sqlstatement)
{
    sql::Statement* stmt = dbcon->createStatement();
    bool bSuccess = false;
    try
    {
        bSuccess = stmt->executeUpdate(sqlstatement);
        dbcon->commit();
    }
```

```
}  
catch (sql::SQLException &e)  
{  
    cout << "ERROR: " << endl;  
    cout << e.what() << endl;  
    cout << e.getErrorCode() << endl;  
    cout << e.getSQLState() << endl;  
}  
delete stmt;  
return bSuccess;  
}  
  
void CommitFileWithOneVersion(string path)  
{  
    FileRecord newFile;  
    bool bSuccess = newFile.CreateFile(path, "initial version");  
  
    string logMessage = "The result was: ";  
    if(bSuccess)  
    {  
        logMessage += "Success!";  
    }  
    else  
    {  
        logMessage += "The nixon snake has failed";  
    }  
  
    log(logMessage);  
}  
  
void CommitFileWithTwoVersions()  
{  
    FileRecord newFile;  
    string path = "testData/testFile.dat";  
    createFile(25, path, 200000);  
    bool bSuccess = newFile.CreateFile(path, "initial version");  
  
    //check that we can't add the same version  
    if(bSuccess)  
    {  
        log("Trying to add same version");  
        bSuccess = !newFile.AddNewVersion(path, "same version");  
        if(bSuccess == false)  
        {  
            log("ERROR: was able to add the same version");  
        }  
    }  
  
    //check that adding a new version works  
    if(bSuccess)  
    {  
        log("Trying to commit a new version");  
        appendFile(25, path, 200);  
    }  
}
```

```
bSuccess = newFile.AddNewVersion(path, "Version 2");
}

string logMessage = "The result was: ";
if(bSuccess)
{
    logMessage += "Success!";
}
else
{
    logMessage += "The nixon snake has failed";
}

log(logMessage);
}

bool GenerateFilesAndCommitVersionsAndVerifyRetrieval(std::string path, unsigned int size,
unsigned int numVersions)
{
    FileRecord myRecord;
    bool bSuccess = true;

    for(unsigned int i = 0; i < numVersions; i++)
    {
        if(bSuccess == false)
        {
            break;
        }
        string currentpath = path + "." + boost::lexical_cast<string>(i);
        createFile(i * 200, currentpath, size);
        if(i == 0)
        {
            bSuccess = myRecord.CreateFile(currentpath, "initial version");
            if(bSuccess == false)
            {
                log("ERROR: Failed to create new file: " + currentpath);
            }
        }
        else
        {
            log("Trying to add new version");
            if(myRecord.IsValid())
            {
                bSuccess = myRecord.AddNewVersion(currentpath, "Version: " + boost::lexical_cast<string>(i) );
                if(bSuccess == false)
                {
                    log("ERROR: Failed to create new version: " + currentpath);
                }
            }
        }
    }
}
```

```
if(bSuccess)
{
string originalName = path + ".0";
for(unsigned int i = 0; i < numVersions; i++)
{
string currentpath = path + "." + boost::lexical_cast<string>(i) + ".ret";
VersionRecord currentRecord(originalName, i + 1);
unsigned int retrievedHash = 0;

if(currentRecord.IsValid())
{
currentRecord.GetFileData(currentpath);
MurmurHash3_x86_32_FromFile(currentpath, MURMUR_SEED_1, &retrievedHash);
if(retrievedHash != currentRecord.GetHash())
{
log("HASHES DID NOT MATCH! " + currentpath);
bSuccess = false;
break;
}
else
{
log("Hashes matched. Retrieval worked correctly");
}
}
else
{
bSuccess = false;
break;
}
}

return bSuccess;
}

void RunTestCommitFileOneVersion()
{
//DropTables();
//CreateTables();
CommitFileWithOneVersion("MurmurHash3.cpp");
}

void RunTestCommitFileWithTwoVersionsRetrieveBoth()
{
//DropTables();
//CreateTables();

string path = "testData/testFile.dat";
string original = "testData/testFile.dat.orig";
string retrievedLatest = "testData/retrievedLatest.dat";
string retrievedOriginal = "testData/retrievedOriginal.dat";

CommitFileWithTwoVersions();
```

```
createFile(25, path + ".orig", 200000);

VersionRecord originalVersion(path, 1);
originalVersion.GetFileData(retrievedOriginal);

VersionRecord latestVersion(path, 2);
latestVersion.GetFileData(retrievedLatest);

unsigned int hash1 = 0;
unsigned int hash2 = 0;
unsigned int hash3 = 0;
unsigned int hash4 = 0;
MurmurHash3_x86_32_FromFile(path, MURMUR_SEED_1, &hash1);
MurmurHash3_x86_32_FromFile(original, MURMUR_SEED_1, &hash2);
MurmurHash3_x86_32_FromFile(retrievedLatest, MURMUR_SEED_1, &hash3);
MurmurHash3_x86_32_FromFile(retrievedOriginal, MURMUR_SEED_1, &hash4);

if(hash1 == hash3 && hash2 == hash4)
{
log("Hashes match. File successfully retrieved");
}
else
{
log("Error. Hashes do not match. File not retrieved correctly");
}
}

void RunTestCommitFileOneVersionRetrieve()
{
//DropTables();
//CreateTables();

/*
string fileinpath = "MurmurHash3.cpp";
string fileoutpath = "MurmurHash3.cpp.ret";
*/
string fileinpath = "testData/nixon.jpg";
string fileoutpath = "testData/nixonout.jpg";
//*/

CommitFileWithOneVersion(fileinpath);
VersionRecord newRec(fileinpath, 1);
newRec.GetFileData(fileoutpath);

unsigned int hash1 = 0;
unsigned int hash2 = 0;
MurmurHash3_x86_32_FromFile(fileinpath, MURMUR_SEED_1, &hash1);
MurmurHash3_x86_32_FromFile(fileoutpath, MURMUR_SEED_1, &hash2);

if(hash1 == hash2)
```

```
{
log("Hashes match. File successfully retrieved");
}
else
{
log("Error. Hashes do not match. File not retrieved correctly");
}
}

void RunTestPurge()
{
    string path = "testData/testFile.dat";
    FileRecord existingFile(path);

    if (!existingFile.IsValid())
    {
        log("FileRecord is not valid.");
        return;
    }

    log("Purging file.");
    existingFile.PurgeOldVersions(0);

    return;
}
```

```

/*
 * File:    main.cpp
 * Author:  philipedwards
 *
 * Created on 16 September 2015, 11:53 AM
 */

#include <QApplication>
#include "ProjectConstants.h"
#include "FileRecord.h"
#include "FileLib.h"

#include <string>

#include "TestUtilities.h"
#include "FileArchiver.h"
#include "boost/lexical_cast.hpp"

//main window header
#include "MyWindow.h"

using namespace std;

int main(int argc, char *argv[]) {
// initialize resources, if needed
// Q_INIT_RESOURCE(resfile);

QApplication app(argc, argv);

        FileLib::SetupWorkingDirectories();
//DropTables();
//CreateTables();
//GenerateFilesAndCommitVersionsAndVerifyRetrieval("testData/testDataagain.dat", 20000,100);

        /*

RunTestCommitFileOneVersionRetrieve();
//RunTestCommitFileWithTwoVersionsRetrieveBoth();

FileArchiver test;
vector<FileRecord> files = test.GetFiles();

for(unsigned int i = 0; i < files.size(); i++)
{
log("File found in database: " + files[i].GetFilename() + " " +
boost::lexical_cast<string>(files[i].GetCurrentVersionNumber()));
}

        */

//RunTestPurge();

// create and show your widgets here

```

```
        MyWindow win;  
        win.show();  
return app.exec();  
  
//return 0;  
}
```



```
#include "ProjectConstants.h"

void logToFile(std::string message)
{
#ifdef DEBUG_LOG_TO_FILE
std::ofstream outFile("debuglog.log", std::ios::app);
if(outFile.is_open())
{
outFile << message << std::endl;
outFile.close();
}
#endif // DEBUG_LOG_TO_FILE
}

void log(std::string message)
{
#ifdef DEBUG_LOGGING
std::cout << message << std::endl;
logToFile(message);
#endif //DEBUG_LOGGING
}
```

```

#ifndef VERSIONRECORD_H
#define VERSIONRECORD_H

#include <string>
class VersionRecord
{
public:

// Constructor
VersionRecord();

// Constructor
VersionRecord(std::string filename, unsigned int versionNumber);

// Destructor
~VersionRecord();

// Returns the id of a version
unsigned int GetVersionId();

// Returns the version number
unsigned int GetVersionNumber();

// Returns the size
unsigned int GetSize();

// Returns the hash of the
unsigned int GetHash();

// Returns the modification time.
unsigned int GetModificationTime();

// Returns a formatted string of the modification time/date.
std::string GetFormattedModificationTime();

//public members for transfer of record to/from persistent storage - the function signatures
will depend on the persistance mechanism that is chosen
bool CreateVersion(std::string keyFilename, std::string pathFilename, unsigned int
currentVersion, unsigned int newHash, std::string newComment);

// Returns the comment on the version
std::string GetComment();

// Returns true if the data in Version is usable
bool IsValid();

bool GetFileData(std::string fileOutPath);

void PurgeVersion();

unsigned int RetrieveSizeFromDisk(std::string path);

protected:

```

```
//retrieves the record information from the database
bool RetrieveVersionRecordFromDB(std::string inFilename, unsigned int versionNumber);
//updates the FileRecord in the database;
bool UpdateRecordInDB();
void Init();

bool InsertVersionIntoDB(std::string keyFileName);

bool InsertBlocks(std::string zipPath);

//The version identifier - generated primary key
unsigned int VersionID;

std::string Filename;

//The version number of this version
unsigned int VersionNumber;
//the length of this version in bytes
unsigned int Size;

unsigned int Time;
unsigned int FileModificationTime;
std::string Comment;

//the has of the entire version of the file
unsigned int Hash;
//database connection
sql::Connection* dbcon;

bool bIsValid;

private:

};

#endif
```

```
#include <iostream>
#include <fstream>

#include "mysql_connection.h"
#include "mysql_driver.h"

#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>
#include <cppconn/prepared_statement.h>

#include "VersionRecord.h"
#include "ProjectConstants.h"
#include "DBConnector.h"
#include "MurmurHash3.h"
#include "boost/lexical_cast.hpp"

#include <streambuf>
#include <istream>
#include <ctime>

#include "CompressUtils.h"
#include "FileLib.h"

using namespace std;

//this struct is copied from this forum post:
// http://stackoverflow.com/a/7782037
struct membuf : std::streambuf
{
    membuf(char* begin, char* end)
    {
        this->setg(begin, begin, end);
    }
};

// Constructor, prepares for use
VersionRecord::VersionRecord()
{
    Init();
}

// Destructor, sets poit for connection to null
VersionRecord::~VersionRecord()
{
    dbcon = NULL;
}

// Constructor, tries to retrieve a record from the db
VersionRecord::VersionRecord(std::string filename, unsigned int versionNumber)
```

```
{
Init();
RetrieveVersionRecordFromDB(filename, versionNumber);
}

// Sets class members to initial values
void VersionRecord::Init()
{
dbcon = DBConnector::GetConnection();
VersionID = 0;
VersionNumber = 0;
Size = 0;
Time = 0;
FileModificationTime = 0;
Hash = 0;
bIsValid = false;
}

// Tries to retrieve a version record from the database
bool VersionRecord::RetrieveVersionRecordFromDB(std::string inFilename, unsigned int
versionNumber)
{
// Prepare for failure
bIsValid = false;

// Setup statement
sql::PreparedStatement *pstmt = dbcon->prepareStatement("select * from Version where filename=?
and version=?");
sql::ResultSet *rs;

// Try to run Query
try
{
pstmt->setString(1,inFilename);
pstmt->setInt(2,versionNumber);
rs = pstmt->executeQuery();

// Output Results
while(rs->next())
{
//count = rs->getUInt(1);
VersionID = rs->getUInt("id");
Filename = rs->getString("filename");
VersionNumber = rs->getUInt("version");
//TODO: change to getInt64?
Size = rs->getUInt("size");
Time = rs->getUInt("time");
FileModificationTime = rs->getUInt("filemodtime");
Comment = rs->getString("comment");
Hash = rs->getUInt("hash");
bIsValid = true;
}
}
```

```

}
catch (sql::SQLException &e)
{
log("ERROR: ");
log(e.what());
log(e.getErrorCode());
log(e.getSQLState());
}

// Clean up
delete rs;
delete pstmt;
rs = NULL;
pstmt = NULL;

// Return result
return bIsValid;
}

// Updates a version record in the db
bool VersionRecord::UpdateRecordInDB()
{
sql::Statement *stmt = dbcon->createStatement();
bool bSuccess = true;

if(stmt == NULL)
{
//failed to get a connection to the database
bSuccess = false;
}

if(IsValid() == false)
{
bSuccess = false;
}

try
{
//create file record
if(bSuccess)
{
//beginning of statement
string sqlstatement = "update Version set ";
//curhash
sqlstatement += "filename = \"" + Filename + "\", ";
sqlstatement += "size = " + boost::lexical_cast<string>(Size) + ", ";
sqlstatement += "time = " + boost::lexical_cast<string>(Time) + ", ";
sqlstatement += "filemodtime = " + boost::lexical_cast<string>(FileModificationTime) + ", ";
sqlstatement += "comment = \"" + Comment + "\", ";
sqlstatement += "version = " + boost::lexical_cast<string>(VersionNumber) + ", ";
sqlstatement += "hash = " + boost::lexical_cast<string>(Hash);
//end of statement

```

```
sqlstatement += " where id = " + boost::lexical_cast<string>(VersionID) + ";;";

log(sqlstatement);

bSuccess = stmt->executeUpdate(sqlstatement);
}
}

catch (sql::SQLException &e)
{
log("ERROR: ");
log(e.what());
log(e.getErrorCode());
log(e.getSQLState());
}

delete stmt;
return bSuccess;
}

unsigned int VersionRecord::GetVersionId()
{
return VersionID;
}

unsigned int VersionRecord::GetVersionNumber()
{
return VersionNumber;
}

unsigned int VersionRecord::GetSize()
{
return Size;
}

unsigned int VersionRecord::GetHash()
{
return Hash;
}

unsigned int VersionRecord::GetModificationTime()
{
return FileModificationTime;
}

string VersionRecord::GetFormattedModificationTime()
{
char buffer[80];
time_t fileTime = FileModificationTime;

strftime(buffer, 80, "%F %T", localtime(&fileTime));
return string(buffer);
}
```

```

bool VersionRecord::InsertVersionIntoDB(string keyFilename)
{
    // Prepare for sql statements
    bool bSuccess = false;
    const char* insertVersion = "insert into Version(filename, version, hash, filemtime,
size, time, comment) values (?, ?, ?, ?, ?, ?, ?)";
    const char* selectVersion = "select id from Version where hash=?";
    sql::PreparedStatement *pstmt = NULL;
    sql::ResultSet *rs = NULL;

    // Try to insert record and get the id
    try
    {
        pstmt = dbcon->prepareStatement(insertVersion);
        pstmt->setString(1, keyFilename);
        pstmt->setInt64(2, VersionNumber);
        pstmt->setInt64(3, Hash);
        pstmt->setInt64(4, FileModificationTime);
        pstmt->setInt64(5, Size);
        pstmt->setInt64(6, Time);
        pstmt->setString(7, Comment);

        bool bNewVersionMade = pstmt->executeUpdate();

        if (bNewVersionMade == false)
        {
            return false;
        }

        // Run Query
        pstmt = dbcon->prepareStatement(selectVersion);
        pstmt->setInt64(1, Hash);
        rs = pstmt->executeQuery();

        // Output Results
        while(rs->next())
        {
            VersionID = rs->getUInt(1);
        }
        bSuccess = true;

        if(bSuccess)
        {
            RetrieveVersionRecordFromDB(keyFilename, VersionNumber);
            if(!IsValid())
            {
                log("Failed to retrieve version record from database. This
version was not created correctly");
                bSuccess = false;
            }
        }
    }
}

```



```
        catch (sql::SQLException &e)
        {
            log("ERROR: ");
            log(e.what());
            log(e.getErrorCode());
            log(e.getSQLState());
            log("Failed to create version in table Version");
            bSuccess = false;
        }

        pstmt->close();
        pstmt = NULL;
        delete rs;
        rs = NULL;
        return bSuccess;
}

bool VersionRecord::CreateVersion(string keyFilename, string pathFilename, unsigned int
currentVersion, unsigned int newHash, string newComment)
{
    bool bSuccess = true;

    // Set all variables to instance of class
    FileModificationTime = FileLib::GetModifiedDate(pathFilename);
    Time = time(0);
    Size = RetrieveSizeFromDisk(pathFilename);
    Comment = newComment;
    VersionNumber = currentVersion;
    Hash = newHash;

    // Create a new version
    bSuccess = InsertVersionIntoDB(keyFilename);

    if(bSuccess)
    {
        // Clean temp folder just incase
        zipRemoveZip();

        // Copy file to temp folder
        zipCopyContents(pathFilename);

        // Compress file in temp folder
        zipCompress();

        // Create Zip path
        string zipPath = "./temp/data.gz";

        // Create Blocks In Database
        bSuccess = InsertBlocks(zipPath);
        if(!bSuccess)
        {
            log("Failed to insert blocks for this version");
        }
    }
}
```

```
// Update
UpdateRecordInDB();

// Clean up temp folder
zipRemoveZip();
}
else
{
log("Failed to create a record in the database for this version");
}

return bSuccess;
}

bool VersionRecord::InsertBlocks(string zipPath)
{
bool bSuccess = true;

// Open File
ifstream ins(zipPath.c_str());
if (!ins.good())
{
log("Failed to open file. Cannot create version");
bSuccess = false;
}

sql::Statement *stmt = dbcon->createStatement();
if(bSuccess)
{
ins.seekg(0,ios::end);
int bytesRemaining = ins.tellg();
ins.seekg(0,ios::beg);

try
{
unsigned int versionIndex = 0;
char block[BLOCK_SIZE];
while (!ins.eof() && bytesRemaining > 0)
{
// Get Block
int blockSize = 0;
if(bytesRemaining > BLOCK_SIZE)
{
blockSize = BLOCK_SIZE;
}
else
{
blockSize = bytesRemaining;
}
bytesRemaining -= blockSize;
ins.read((char*)block, blockSize);
log("block size is " + boost::lexical_cast<string>(blockSize));
}
```

```
if (blockSize == 0)
{
break;
}

// Hash 1
unsigned int hash1 = 0;
MurmurHash3_x86_32(block, blockSize, MURMUR_SEED_1 , &hash1);

// Hash 2
unsigned int hash2 = 0;
MurmurHash3_x86_32(block, blockSize, MURMUR_SEED_2 , &hash2);

// Query DB Hash 1 in table Blocks
unsigned int blockId = 0;

// Run Query
sql::ResultSet *rs = stmt->executeQuery("select id from Block where hash1 = " +
boost::lexical_cast<string>(hash1));

// Output Results
while(rs->next())
{
blockId = rs->getUInt(1);
}

delete rs;

// If hash 1 already exists
if (blockId != 0)
{
log("Hash is not equal to zero");
// Query DB Hash 2 in table Blocks
unsigned int result;

// Run Query
sql::ResultSet *rs = stmt->executeQuery("select id from Block where hash2 = " +
boost::lexical_cast<string>(hash2) + " and id = " + boost::lexical_cast<string>(blockId));

// Output Results
while(rs->next())
{
result = rs->getUInt(1);
}

delete rs;

// If hash 2 matches the same id as
if (result != 0)
{
// Use existing block
bSuccess = stmt->executeUpdate("insert into VtoB(versionid, blockid, versionindex) values (" +
boost::lexical_cast<string>(this->VersionID) + ", " + boost::lexical_cast<string>(blockId) + ",
```

```

" + boost::lexical_cast<string>(versionIndex++) + ")");
}
else
{
// Create a new block
sql::PreparedStatement *pstmt = dbcon->prepareStatement("insert into Block(hash1, hash2, data)
values (?, ?, ?)");
pstmt->setUInt(1, hash1);
pstmt->setUInt(2, hash2);
membuf sbuf(block, block + blockSize);
istream in(&sbuf);
pstmt->setBlob(3, &in);
bSuccess = pstmt->executeUpdate();
delete pstmt;
bSuccess = stmt->executeUpdate("commit");

// Link block with VtoB
bSuccess = stmt->executeUpdate("insert into VtoB(versionid, blockid, versionindex) values (" +
boost::lexical_cast<string>(VersionID) + ", " + boost::lexical_cast<string>(blockId) + ", " +
boost::lexical_cast<string>(versionIndex++) + ")");
dbcon->commit();
}
}
else
{
log("Hash equals zero ");
// Create a new block
sql::PreparedStatement *pstmt = dbcon->prepareStatement("insert into Block(hash1, hash2, data)
values (?, ?, ?)");
pstmt->setUInt(1, hash1);
pstmt->setUInt(2, hash2);
membuf sbuf(block, block + blockSize);
istream in(&sbuf);
pstmt->setBlob(3, &in);
bSuccess = pstmt->executeUpdate();
delete pstmt;

// Run Query
dbcon->commit();
int i = 0;
bool bFound = false;
while (bFound == false && i < 100)
{
i++;
string sqlstatement = "select id from Block where hash1 = " + boost::lexical_cast<string>(hash1)
+ " and hash2 = " + boost::lexical_cast<string>(hash2);
log(sqlstatement);
sql::ResultSet *rs1 = stmt->executeQuery(sqlstatement);

if(rs1->next() == false)
{
log("Failed to find block that was just committed");
bSuccess = false;
}
}
}
}

```

```

}
else
{
    bSuccess = true;
    bFound = true;
    blockId = rs1->getUInt(1);
}
delete rs1;
}

if(bSuccess)
{
    log("blockId = " + boost::lexical_cast<string>(blockId));

    // Link block with VtoB
    string sqlstatement = "insert into VtoB(versionid, blockid, versionindex) values (" +
        boost::lexical_cast<string>(VersionID) + ", " + boost::lexical_cast<string>(blockId) + ", " +
        boost::lexical_cast<string>(versionIndex++) + ")";
    log(sqlstatement);
    bSuccess = stmt->executeUpdate(sqlstatement);
    dbcon->commit();
}
}
}
}

catch (sql::SQLException &e)
{
    log("ERROR: ");
    log(e.what());
    log(e.getErrorCode());
    log(e.getSQLState());
    bSuccess = false;
}
}

ins.close();

delete stmt;
stmt = NULL;

return bSuccess;
}

// Returns the size of a file passed to it
unsigned int VersionRecord::RetrieveSizeFromDisk(string path)
{
    ifstream ins(path.c_str());

    if (!ins.good())
    {
        log("Failed to open file. Cannot get file size.");
    }
}

```

```

}

int fileSize = 0;
if(ins.good())
{
ins.seekg (0, ios::end);
fileSize = ins.tellg();
ins.seekg (0, ios::beg);
}

return fileSize;
}

std::string VersionRecord::GetComment()
{
return Comment;
}

bool VersionRecord::IsValid()
{
return bIsValid;
}

bool VersionRecord::GetFileData(std::string fileOutPath)
{
bool bSuccess = IsValid();
sql::Statement *stmt = dbcon->createStatement();

if(stmt == NULL)
{
//failed to get a connection to the database
bSuccess = false;
}

ofstream outFile("./temp/data.gz");

if(outFile.is_open() == false)
{
log("Unable to open file to write on disk");
bSuccess = false;
}

try
{
//create file record
if(bSuccess)
{
string sqlstatement = "select blockid from VtoB where versionid = " +
boost::lexical_cast<string>(VersionID) + " order by versionindex ASC";

log(sqlstatement);

sql::ResultSet *rs = stmt->executeQuery(sqlstatement);

```

```
int blocksRetrieved = 0;
while(rs->next() && bSuccess)
{
    blocksRetrieved++;
    log("retrieved block");

    unsigned int blockid = rs->getUInt("blockid");

    //for all block records, fetch block, write to disk
    string blockretsql = "select data from Block where id = " +
        boost::lexical_cast<string>(blockid);
    sql::ResultSet *rs2 = stmt->executeQuery(blockretsql);

    if(rs2->next())
    {
        istream* data = rs2->getBlob("data");
        char outbuf[BLOCK_SIZE];
        int blobsize = data->readsome(outbuf, BLOCK_SIZE);
        outFile.write(outbuf, blobsize);
    }
    else
    {
        bSuccess = false;
    }

    delete rs2;
    rs2 = NULL;
}

if(blocksRetrieved == 0)
{
    bSuccess = false;
}

delete rs;
rs = NULL;
}

catch (sql::SQLException &e)
{
    log("ERROR: ");
    log(e.what());
    log(e.getErrorCode());
    log(e.getSQLState());
    bSuccess = false;
}

outFile.close();

zipUncompressTo(fileOutPath);
```

```
delete stmt;
stmt = NULL;
return bSuccess;
}

void VersionRecord::PurgeVersion()
{
    // Catch invalid version
    if (!bIsValid)
        return;

    sql::Statement *stmt = dbcon->createStatement();

    try
    {
        // Delete records from VtoB
        stmt->executeUpdate("delete from VtoB where versionid = " +
boost::lexical_cast<string>(VersionID));

        // Clean data Blocks up
        // This will show which blocks need to be deleted "select id from Block where id not in
(select b.id from Block b join VtoB v on b.id = v.blockid);"
        stmt->executeUpdate("delete b from Block b left join VtoB v on v.blockid = b.id where
v.blockid is null");

        // Remove the version record
        stmt->executeUpdate("delete from Version where id = " +
boost::lexical_cast<string>(VersionID));

    }
    catch (sql::SQLException e)
    {
        log("ERROR: ");
        log(e.what());
        log(e.getErrorCode());
        log(e.getSQLState());
    }

    delete stmt;

    // Record no longer exists, it becomes invalid
    bIsValid = false;
}
```



```
/*
 * File:    backendtests.h
 * Author:  philipedwards
 *
 * Created on 30/09/2015, 6:40:09 PM
 */

#ifndef BACKENDTESTS_H
#define BACKENDTESTS_H

#include <cppunit/extensions/HelperMacros.h>

class backendtests : public CPPUNIT_NS::TestFixture {
    CPPUNIT_TEST_SUITE(backendtests);

    //CPPUNIT_TEST(testMethod);
    //CPPUNIT_TEST(testFailedMethod);
    CPPUNIT_TEST(hashFileTest);
    CPPUNIT_TEST(commitRetrieveTest);
    CPPUNIT_TEST(purgeTest);

    CPPUNIT_TEST_SUITE_END();

public:
    backendtests();
    virtual ~backendtests();
    void setUp();
    void tearDown();

private:
    //void testMethod();
    //void testFailedMethod();

    void hashFileTest();
    void commitRetrieveTest();
    void purgeTest();

    bool GenerateFilesAndCommitVersionsAndVerifyRetrieval(std::string path, unsigned int size,
unsigned int numVersions);
};

#endif/* BACKENDTESTS_H */
```

```
/*
 * File:    backendtests.cpp
 * Author:  philipedwards
 *
 * Created on 30/09/2015, 6:40:09 PM
 */

#include "backendtests.h"
#include "MurmurHash3.h"
#include "ProjectConstants.h"
#include "TestUtilities.h"
#include "FileArchiver.h"
#include "FileRecord.h"
#include "VersionRecord.h"
#include "FileLib.h"
#include "boost/lexical_cast.hpp"

#include <string>
using namespace std;

CPPUNIT_TEST_SUITE_REGISTRATION(backendtests);

backendtests::backendtests()
{
}

backendtests::~~backendtests()
{
}

void backendtests::setUp()
{
    FileLib::SetupWorkingDirectories();
    DropTables();
    CreateTables();
}

void backendtests::tearDown()
{
}

bool backendtests::GenerateFilesAndCommitVersionsAndVerifyRetrieval(std::string path, unsigned
int size, unsigned int numVersions)
{
    FileRecord myRecord;
    bool bSuccess = true;

    for(unsigned int i = 0; i < numVersions; i++)
    {
        if(bSuccess == false)
        {
            break;
        }
    }
}
```

```

}

string currentpath = path + "." + boost::lexical_cast<string>(i);
createFile(i * 200, currentpath, size);
if(i == 0)
{
bSuccess = myRecord.CreateFile(currentpath, "initial version");
if(bSuccess == false)
{
log("ERROR: Failed to create new file: " + currentpath);
CPPUNIT_ASSERT_MESSAGE("ERROR: Failed to create new file: " + currentpath,false);
}
}
else
{
log("Trying to add new version");
if(myRecord.IsValid())
{
bSuccess = myRecord.AddNewVersion(currentpath, "Version: " + boost::lexical_cast<string>(i) );
if(bSuccess == false)
{
log("ERROR: Failed to create new version: " + currentpath);
CPPUNIT_ASSERT_MESSAGE("ERROR: Failed to create new version: " + currentpath,false);
}
}
}

}

if(bSuccess)
{
string originalName = path + ".0";
for(unsigned int i = 0; i < numVersions; i++)
{
string currentpath = path + "." + boost::lexical_cast<string>(i) + ".ret";
VersionRecord currentRecord(originalName, i + 1);
unsigned int retrievedHash = 0;

if(currentRecord.IsValid())
{
currentRecord.GetFileData(currentpath);
MurmurHash3_x86_32_FromFile(currentpath, MURMUR_SEED_1, &retrievedHash);
if(retrievedHash != currentRecord.GetHash())
{
log("HASHES DID NOT MATCH! " + currentpath);
CPPUNIT_ASSERT_MESSAGE("HASHES DID NOT MATCH! " + currentpath,false);
bSuccess = false;
break;
}
else
{
log("Hashes matched. Retrieval worked correctly");
}
}
}
}

```

```
else
{
bSuccess = false;
CPPUNIT_ASSERT_MESSAGE("Invalid version record retrieved",false);
break;
}
}
}

return bSuccess;
}

void backendtests::hashFileTest()
{
string file1 = "testData/hashFile1.dat";
string file2 = "testData/hashFile2.dat";

//hash files of different lengths

createFile(24000, file1, 14000);
createFile(2888000, file2, 24000);

unsigned int hash1 = 0;
unsigned int hash2 = 0;

MurmurHash3_x86_32_FromFile(file1, MURMUR_SEED_1, &hash1);
MurmurHash3_x86_32_FromFile(file2, MURMUR_SEED_1, &hash2);

//if the hashes match this is an error
if(hash1 == hash2)
{
CPPUNIT_ASSERT_MESSAGE("Files of different lengths", false);
}

//hash files of the same length
createFile(24000, file1, 14000);
createFile(28000, file2, 14000);

MurmurHash3_x86_32_FromFile(file1, MURMUR_SEED_1, &hash1);
MurmurHash3_x86_32_FromFile(file2, MURMUR_SEED_1, &hash2);

if(hash1 == hash2)
{
CPPUNIT_ASSERT_MESSAGE("Files of the same length", false);
}

//hash the same file with different seeds
MurmurHash3_x86_32_FromFile(file1, MURMUR_SEED_1, &hash1);
MurmurHash3_x86_32_FromFile(file1, MURMUR_SEED_2, &hash2);
if(hash1 == hash2)
{
CPPUNIT_ASSERT_MESSAGE("Same file with different seeds", false);
}
```

```

}
}

void backendtests::commitRetrieveTest()
{
    bool bSuccess;
    bSuccess = GenerateFilesAndCommitVersionsAndVerifyRetrieval("testData/20VersionFile.dat", 30000,
20);
    CPPUNIT_ASSERT_MESSAGE("20 version file had an unknown error", bSuccess);

    bSuccess = GenerateFilesAndCommitVersionsAndVerifyRetrieval("testData/30VersionFile.dat",
100000, 30);
    CPPUNIT_ASSERT_MESSAGE("30 version file had an unknown error", bSuccess);

    bSuccess = GenerateFilesAndCommitVersionsAndVerifyRetrieval("testData/40VersionFile.dat", 80000,
40);
    CPPUNIT_ASSERT_MESSAGE("40 version file had an unknown error", bSuccess);
}

void backendtests::purgeTest()
{
    std::string path = "testData/purgeFile.dat";
    unsigned int size = 24000;
    unsigned int numVersions = 5, numVersionsToKeep = 2;
    bool bSuccess = GenerateFilesAndCommitVersionsAndVerifyRetrieval(path, size, numVersions);

    // GenerateFilesAndCommitVersionsAndVerifyRetrieval() will check equality of hashes on
generation.
    CPPUNIT_ASSERT(bSuccess);

    FileRecord fileRec(path + ".0");

    CPPUNIT_ASSERT(fileRec.IsValid());
    fileRec.PurgeOldVersions(numVersionsToKeep);

    vector<VersionRecord> versionRecs = fileRec.GetAllVersions();

    // Check that the expected number of versions are the actual number of versions
(post-purge).
    CPPUNIT_ASSERT(numVersionsToKeep == versionRecs.size());

    unsigned int hash;
    std::string testPath = path + ".ret";

    for(vector<VersionRecord>::iterator it = versionRecs.begin(); it != versionRecs.end(); ++it)
    {
        it->GetFileData(testPath);
        MurmurHash3_x86_32_FromFile(testPath, MURMUR_SEED_1, &hash);

        CPPUNIT_ASSERT(hash == it->GetHash());
    }
}

```

```
/*
 * File:    backendtestrunner.cpp
 * Author:  philipedwards
 *
 * Created on 30/09/2015, 6:40:10 PM
 */

#include <cppunit/BriefTestProgressListener.h>
#include <cppunit/CompilerOutputter.h>
#include <cppunit/extensions/TestFactoryRegistry.h>
#include <cppunit/TestResult.h>
#include <cppunit/TestResultCollector.h>
#include <cppunit/TestRunner.h>

int main()
{
    // Create the event manager and test controller
    CPPUNIT_NS::TestResult controller;

    // Add a listener that collects test result
    CPPUNIT_NS::TestResultCollector result;
    controller.addListener(&result);

    // Add a listener that print dots as test run.
    CPPUNIT_NS::BriefTestProgressListener progress;
    controller.addListener(&progress);

    // Add the top suite to the test runner
    CPPUNIT_NS::TestRunner runner;
    runner.addTest(CPPUNIT_NS::TestFactoryRegistry::getRegistry().makeTest());
    runner.run(controller);

    // Print test in a compiler compatible format.
    CPPUNIT_NS::CompilerOutputter outputter(&result, CPPUNIT_NS::stdCOut());
    outputter.write();

    return result.wasSuccessful() ? 0 : 1;
}
```

```
/*
 * File:    FileLibTester.h
 * Author:  jjc224
 *
 * Created on 23/09/2015, 4:24:16 PM
 */

#ifndef FILELIBTESTER_H
#define FILELIBTESTER_H

#include <cppunit/extensions/HelperMacros.h>

class FileLibTester : public CPPUNIT_NS::TestFixture
{
    CPPUNIT_TEST_SUITE(FileLibTester);

    CPPUNIT_TEST(testAppendPath);
    CPPUNIT_TEST(testGetFilename);
    CPPUNIT_TEST(testGetPath);
    CPPUNIT_TEST(testNormalize);
    CPPUNIT_TEST(testSplitPath);

    CPPUNIT_TEST_SUITE_END();

public:
    /*
    FileLibTester();
    virtual ~FileLibTester();
    void setUp();
    void tearDown();
    */

private:
    void testAppendPath();
    void testGetFilename();
    void testGetHash();
    void testGetModifiedDate();
    void testGetPath();
    void testNormalize();
    void testSplitPath();

    // Helper functions (not in FileLib).
    std::string generateUnnormalizedPath(const std::string elements[], const std::size_t SIZE);
};

#endif/* FILELIBTESTER_H */
```

```

/*
 * File:    FileLibTester.cpp
 * Author:  jjc224
 *
 * Created on 23/09/2015, 4:24:17 PM
 */

#include "FileLibTester.h"
#include "FileLib.h"
#include <climits>

CPPUNIT_TEST_SUITE_REGISTRATION(FileLibTester);

void FileLibTester::testAppendPath()
{
    const std::size_t NUM_PATH_ELEMENTS_1 = 2;
    const std::size_t NUM_PATH_ELEMENTS_2 = 3;

    std::string pathElements1[NUM_PATH_ELEMENTS_1] = {"/test", "path1"};
    std::string pathElements2[NUM_PATH_ELEMENTS_2] = {"another", "path2", "test"};

    std::string path1 = generateUnnormalizedPath(pathElements1, NUM_PATH_ELEMENTS_1);
    std::string path2 = generateUnnormalizedPath(pathElements2, NUM_PATH_ELEMENTS_2);
    std::string expectedResult = "/test/path1/another/path2/test/";
    FileLib fileLib;
    std::string result = fileLib.AppendPath(path1, path2);

    CPPUNIT_ASSERT(result == expectedResult);
}

void FileLibTester::testGetFilename()
{
    const std::size_t NUM_PATH_ELEMENTS = 2;
    std::string pathElements[NUM_PATH_ELEMENTS] = {"/some", "path"};

    std::string file = "file.ext";
    std::string path = generateUnnormalizedPath(pathElements, NUM_PATH_ELEMENTS) + file;
    FileLib fileLib;
    std::string result = fileLib.GetFilename(path);
    std::string expectedResult = file;

    CPPUNIT_ASSERT(result == expectedResult);
}

void FileLibTester::testGetPath()
{
    const std::size_t NUM_PATH_ELEMENTS = 5;
    std::string pathElements[NUM_PATH_ELEMENTS] = {"/path", "to", "file.ext"};

    std::string path = generateUnnormalizedPath(pathElements, NUM_PATH_ELEMENTS);
    FileLib fileLib;
    std::string result = fileLib.GetPath(path);
    std::string expectedResult;

```



```

        // This produces a string of the form "/some/path/to/a/file".
        for(std::size_t i = 0; i < NUM_PATH_ELEMENTS - 1; ++i)    // The -1 is so that we don't
include the filename.
            expectedResult += pathElements[i] + "/";

        CPPUNIT_ASSERT(result == expectedResult);
    }

void FileLibTester::testNormalize()
{
    const std::size_t NUM_PATH_ELEMENTS = 5;
    std::string pathElements[NUM_PATH_ELEMENTS] = {"some", "path", "to", "a", "file"};

    std::string path = generateUnnormalizedPath(pathElements, NUM_PATH_ELEMENTS);
    FileLib fileLib;
    std::string result = fileLib.Normalize(path);

    std::string expectedResult;

    // This produces a string of the form "/some/path/to/a/file".
    for(std::size_t i = 0; i < NUM_PATH_ELEMENTS; ++i)
        expectedResult += pathElements[i] + "/";

    CPPUNIT_ASSERT(result == expectedResult);
}

void FileLibTester::testSplitPath()
{
    const std::size_t NUM_PATH_ELEMENTS = 5;
    std::string pathElements[NUM_PATH_ELEMENTS] = {"C:", "some", "test", "path", "file.ext"};
    std::vector<std::string> expectedResult;
    std::string path = generateUnnormalizedPath(pathElements, NUM_PATH_ELEMENTS);

    for(std::size_t i = 0; i < NUM_PATH_ELEMENTS; ++i)
        expectedResult.push_back(pathElements[i]);

    FileLib fileLib;
    std::vector<std::string> result = fileLib.SplitPath(path);

    CPPUNIT_ASSERT(result == expectedResult);
}

// Helper functions (not in FileLib).
// To be tweaked (this is just a quick test).
std::string FileLibTester::generateUnnormalizedPath(const std::string elements[], const
std::size_t SIZE)
{
    std::string path, currPath;
    std::string delimiters[2] = {"\\", "\\\\"};
    const int MAX_DELIMITER_SIZE = 2;

    // This produces a non-normalized path of the form:

```

```
// elements[0]\elements[1]//elements[2]\\elements[3]////...  
for(std::size_t i = 0; i < SIZE; ++i)  
{  
    currPath = elements[i] + std::string(delimiters[(i + 1) % 2]);  
    path.append(currPath);  
}  
  
return path;
```

```
}
```

```
/*
 * File:    CompressUtils.h
 * Author:  thomas
 *
 * Created on 26 September 2015, 4:34 PM
 */

#ifndef COMPRESSUTILS_H
#define COMPRESSUTILS_H

#include <string>

// Creates a copy of a file to the temp folder
void zipCopyContents(std::string path);

// Compresses a file given a path to a file
void zipCompress();

// Uncompresses a zip to the specified location path
void zipUncompressTo(std::string path);

// Removes all zips from the temp folder
void zipRemoveZip();

#endif/* COMPRESSUTILS_H */
```

```
#include <iostream>
#include <cstdlib>
#include <string>
#include "CompressUtils.h"
#include "ProjectConstants.h"

using namespace std;

// VersionRecord::CreateVersion
// VersionRecord::GetFileData

// Compresses a file given a path to a file
void zipCompress()
{
    string command = "gzip -9 ";
    command.append(COMPRESSION_WORK_PATH);
    command.append("data");
    system(command.c_str());
}

// Uncompresses a zip to the specified location path
void zipUncompressTo(string path)
{
    string command = "gunzip -c ";
    command.append(COMPRESSION_WORK_PATH);
    command.append("data.gz > ");
    command.append(path);
    system(command.c_str());
    zipRemoveZip();
}

// Creates a copy of a file to the temp folder
void zipCopyContents(string path)
{
    string command = "cp ";
    command.append(path);
    command.append(" ");
    command.append(COMPRESSION_WORK_PATH);
    command.append("/data");
    system(command.c_str());
}

// Removes all files from temp folder
void zipRemoveZip()
{
    string command = "rm ";
    command.append(COMPRESSION_WORK_PATH);
    command.append("data.gz -f");
    system(command.c_str());
}
```

```
/*
 * File:    DBConnector.h
 * Author:  philipedwards
 *
 * Created on 16 September 2015, 12:36 PM
 */

#ifndef DBCONNECTOR_H
#define DBCONNECTOR_H

#include <mysql_connection.h>
#include <mysql_driver.h>

class DBConnector
{
public:
    DBConnector();
    virtual ~DBConnector();
    static sql::Connection* GetConnection();
private:
};

#endif/* DBCONNECTOR_H */
```

```
/*
 * File:    DBConnector.cpp
 * Author: philipedwards
 *
 * Created on 16 September 2015, 12:36 PM
 */

#include <string>
#include <fstream>

#include <mysql_connection.h>
#include <mysql_driver.h>

#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>

#include "boost/lexical_cast.hpp"

#include "DBConnector.h"
#include "ProjectConstants.h"

using namespace std;

static sql::Connection* dbcon = NULL;
static sql::Driver* driver = NULL;
static bool bInitialised = false;

string decrypt(string s)
{
    for (unsigned int i = 0; i < s.size(); i++)
    {
        s[i] = s[i] - 1 - i%2;
    }

    return s;
}

DBConnector::DBConnector()
{
}

DBConnector::~DBConnector()
{
}

sql::Connection* DBConnector::GetConnection()
{
    //sql::Connection* dbcon = NULL;
```

```
//sql::Driver* driver = NULL;
if(bInitialised != false && dbcon != NULL)
{
return dbcon;
}

// Get data for connection from file
string host;
string dbname;
string user;
string pw;

ifstream ins("db.txt");
getline(ins, dbname);
getline(ins, user);
getline(ins, pw);
getline(ins, host);
ins.close();

/*
host = decrypt(host);
dbname = decrypt(dbname);
user = decrypt(user);
pw = decrypt(pw);

*/

    host = "127.0.0.1";
    dbname = "redsquare";
    user = pw = "root";

log("host: " + host);
log("user: " + user);
log("pw: " + pw);
//*/

// Connect to database
try
{
driver = get_driver_instance();
dbcon = driver->connect(host, user, pw);
dbcon->setSchema(dbname);
bInitialised = true;
}
catch (sql::SQLException&e)
{
log("ERROR: ");
log(e.what());
log(boost::lexical_cast<string>(e.getErrorCode()));
log(e.getSQLState());
}

//dbcon->setTransactionIsolation(sql::TRANSACTION_READ_UNCOMMITTED);
```

```
return dbcon;  
}
```



```
/*
 * File:    FileArchiver.cpp
 * Author:  Thomas Nixon
 *
 * Created on 19 September 2015, 3:59 PM
 */

#ifndef FILEARCHIVER_H
#define FILEARCHIVER_H

#include "mysql_connection.h"
#include "mysql_driver.h"
#include <string>
#include <vector>

#include "FileRecord.h"

class FileArchiver
{
protected:
    sql::Connection* dbcon;
public:
    // Constructor
    FileArchiver();

    // Destructor
    ~FileArchiver();

    // Checks if a file exists already
    bool Exists(std::string filename);

    // Gets a file record for a filename
    FileRecord GetFile(std::string filename);

    // Returns the number of versions on a file
    int GetNumVersions(std::string filename);

    // Adds a new file to the database
    bool AddFile(std::string filename, std::string comment);

    // Adds a new version to a file in the database
    bool AddVersion(std::string filename, std::string comment);

    // Returns all the files in the database as a vector
    std::vector<FileRecord> GetFiles();

    // Returns all the filename stored in the database
    std::vector<std::string> GetFileNames();

    // Detect all file changes and return the ones that have changed
    std::vector<FileRecord> DetectChangedRecords();
};
```

```
#endif
```

```
// File: FileArchiver.cpp
// Author: Thomas Nixon

#include "mysql_connection.h"
#include "mysql_driver.h"

#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>

#include <string>
#include <iostream>
#include <sstream>
#include <fstream>
#include "FileArchiver.h"
#include "DBConnector.h"
#include "FileRecord.h"
#include "VersionRecord.h"
#include "ProjectConstants.h"
#include "boost/lexical_cast.hpp"

#include "MurmurHash3.h"

using namespace std;

// Constructor
FileArchiver::FileArchiver()
{
    dbcon = DBConnector::GetConnection();
}

// Destructor
FileArchiver::~FileArchiver()
{
    //dbcon->close();
    //delete dbcon;
    dbcon = NULL;
}

// Checks if a file exists already
bool FileArchiver::Exists(std::string filename)
{
    FileRecord myFile(filename);
    return myFile.IsValid();
}

// Gets a file record for a filename
FileRecord FileArchiver::GetFile(std::string filename)
{
    FileRecord myFile(filename);
    return myFile;
}
```

```
// Returns the number of versions on a file
int FileArchiver::GetNumVersions(std::string filename)
{
    FileRecord myFile(filename);
    return myFile.GetNumberOfVersions();
}

// Adds a new file to the database
bool FileArchiver::AddFile(string filename, string comment)
{
    FileRecord myFile;
    return myFile.CreateFile(filename, comment);
}

// Adds a new version to a file in the database
bool FileArchiver::AddVersion(string filename, string comment)
{
    FileRecord myFile(filename);
    if(myFile.IsValid())
    {
        return myFile.AddNewVersion(filename, comment);
    }

    return false;
}

// Returns all the files in the database as a vector
vector<FileRecord> FileArchiver::GetFiles()
{
    vector<string> filenames = GetFileNames();
    vector<FileRecord> files;

    for(unsigned int i = 0; i < filenames.size(); i++)
    {
        FileRecord myRec(filenames[i]);
        if(myRec.IsValid())
        {
            files.push_back(myRec);
        }
    }

    return files;
}

// Returns all the filename stored in the database
vector<string> FileArchiver::GetFileNames()
{
    vector<string> filenames;
    string sqlstatement = "select filename from File";

    try
    {
```

```
// Run Query
sql::Statement *stmt = dbcon->createStatement();
sql::ResultSet *rs = stmt->executeQuery(sqlstatement);

// Output Results
while(rs->next())
{
string filename = rs->getString("filename");
filenames.push_back(filename);
}

delete rs;
rs = NULL;
delete stmt;
stmt = NULL;
}
catch (sql::SQLException &e)
{
log("ERROR: ");
log(e.what());
log(boost::lexical_cast<string>(e.getErrorCode()));
log(e.getSQLState());
}

return filenames;
}

// Detect all file changes and return the ones that have changed
vector<FileRecord> FileArchiver::DetectChangedRecords()
{
vector<FileRecord> records = GetFiles();
vector<FileRecord> changed;

for(unsigned int i = 0; i < records.size(); i++)
{
FileRecord& myFile = records[i];
if(myFile.IsValid() && myFile.IsChanged())
{
changed.push_back(myFile);
}
}

return changed;
}
```

```
// Author: Joshua Coleman (jjc224).

#ifndef FILELIB_H
#define FILELIB_H

#include <vector>
#include <string>

class FileLib
{
public:
static std::vector<std::string> SplitPath(std::string path);
static std::string Normalize(std::string path);
static std::string GetPath(std::string path);
static std::string GetFilename(std::string path);
static time_t GetModifiedDate(std::string path);           // Returns modification
date in seconds.
        static std::string GetFormattedModificationDate(std::string path);    //
Returns a user-friendly date for the GUI.
static std::string AppendPath(std::string &path1, std::string path2);    // Appends path2 onto
path1.
static unsigned int GetHash(std::string path);
        static void SetupWorkingDirectories();
};

#endif/* FILELIB_H */
```

```
// Author: Joshua Coleman (jjc224).

#include "FileLib.h"
#include "MurmurHash3.h"
#include "ProjectConstants.h"
#include <cstdlib> // For
FileLib::SetupWorkingDirectories(): command execution (system()).
#include <boost/algorithm/string/classification.hpp> // For FileLib::SplitPath(): string
searching (boost::is_any_of()) to assist splitting.
#include <boost/algorithm/string/split.hpp> // For FileLib::SplitPath(): string splitting
(boost::split()).
#include <boost/algorithm/string.hpp> // For FileLib::Normalize(): string trimming
(boost::algorithm::trim()).
#include <boost/regex.hpp> // For FileLib::Normalize(): substring
replacement through regular expressions.
#include <boost/filesystem.hpp> // For FileLib::GetModificationDate():
uses boost::filesystem::last_write_time() to obtain modification date.

using namespace std;

vector<string> FileLib::SplitPath(string path)
{
vector<string> splittedPath;

path = Normalize(path);
boost::split(splittedPath, path, boost::is_any_of("/"), boost::token_compress_on); // Split
path by forward slash ("/") into vector.

// There will be a null-byte in the last string if the path ends in the delimiter.
// So, remove it.
if(splittedPath.back().empty())
{
splittedPath.pop_back();
}

return splittedPath;
}

// Replaces one or more backslashes and two or more forward slashes with a single forward slash.
string FileLib::Normalize(string path)
{
boost::regex re("\\\\+|//+");
path = boost::regex_replace(path, re, "/", boost::match_default | boost::format_all);

return path;
}

string FileLib::GetPath(string path)
{
path = Normalize(path);
size_t lastSlashIndex = path.find_last_of("/");

if(lastSlashIndex != string::npos)
```

```

{
path = path.substr(0, lastSlashIndex);    // Whatever is up to the final slash must be the path.
}

return path;
}

string FileLib::GetFilename(string path)
{
return SplitPath(path).back();
}

time_t FileLib::GetModifiedDate(string path)
{
time_t modificationTime;
path = Normalize(path);

try
{
modificationTime = boost::filesystem::last_write_time(path.c_str());
}
catch(const boost::filesystem::filesystem_error &e)
{
cerr << "FileLib::GetModifiedDate(): error accessing file '" << path << "' (" <<
e.code().message() << ")." << endl;
}

return modificationTime;
}

string FileLib::GetFormattedModificationDate(string path)
{
    char buffer[80];
    time_t fileTime = GetModifiedDate(path);

    strftime(buffer, 80, "%F %T", localtime(&fileTime));
    return string(buffer);
}

string FileLib::AppendPath(string &path1, string path2)
{
{
path1.append(path2);
    path1 = Normalize(path1);

    return path1;
}

unsigned int FileLib::GetHash(string path)
{
{
unsigned int hash;

MurmurHash3_x86_32_FromFile(path, MURMUR_SEED_1, &hash);
return hash;
}
}

```



```
}

void FileLib::SetupWorkingDirectories()
{
    const string unitTestPath      = "./testData";
    const string tempRetrievalPath = "./temp";

    const string clearUnitTestPath      = "rm -rf " + unitTestPath;
    const string clearTempRetrievalPath = "rm -rf " + tempRetrievalPath;
    const string createUnitTestPath     = "mkdir  " + unitTestPath;
    const string createTempRetrievalPath = "mkdir  " + tempRetrievalPath;

    system(clearUnitTestPath.c_str());
    system(clearTempRetrievalPath.c_str());
    system(createUnitTestPath.c_str());
    system(createTempRetrievalPath.c_str());
}
```

```
#ifndef FILERECORD_H
#define FILERECORD_H

#include <string>
#include <vector>
#include "DBConnector.h"

#include "VersionRecord.h"

class FileRecord
{
public:
FileRecord();
~FileRecord();
FileRecord(std::string filename);

//Creates the file record on the data
bool CreateFile(std::string filename, std::string newComment);

VersionRecord GetVersion(unsigned int versionNum);

std::vector<VersionRecord> GetAllVersions();

void PurgeOldVersions(int numberOfVersionsToKeep);

// Returns the number of versions this file has
int GetNumberOfVersions();

unsigned int GetCurrentVersionNumber();

//Ensures there is a valid corresponding record in the database
bool IsValid();

bool GetVersionFileContents(unsigned int versionNumber, std::string fileOutPath);

// Gets the full file path
std::string GetFilename();

// Gets the length of a specific version of the file
unsigned int GetVersionSize(unsigned int versionNumber);

// Friendly function for adding a new file version
// Returns false if the version has not changed
bool AddNewVersion(std::string NewFileVersionPath, std::string newComment);

//returns true if the file on disk has been modified
//returns false if the file is the same
//returns false if the file does not exist on disk
bool IsChanged();

protected:
//retrieves the record information from the database
```

```
bool RetrieveFileRecordFromDB(std::string inFilename);
//called by constructors to perform common functionality
void Init();
//updates the FileRecord in the database;
bool UpdateRecordInDB();

// The full path name of the file
std::string Filename;

// The hash of the current version of the file
unsigned int CurrentVersionHash;

// The current revision number
unsigned int CurrentVersion;

// the number of versions this file has had
unsigned int NumberOfVersions;

// The last modified time of the file
int ModifiedTime;

// The primary ID of the file in the datastore
unsigned int FileID;

// Adds a new version of the file
// Will fail if the hash of the filebuffer data is the same as the current version hash

//uses murmur3 to get a 32 bit hash of the full file
unsigned int GetHashOfFileBuffer(int FileLength, const char* FileBuffer);

sql::Connection* dbcon;

bool bIsValid;
};

#endif
```

```
#include "FileRecord.h"
#include "MurmurHash3.h"
#include <stdint.h>

#include <fstream>

#include <cppconn/driver.h>
#include <cppconn/exception.h>
#include <cppconn/resultset.h>
#include <cppconn/statement.h>

#include <boost/filesystem.hpp>

#include "DBConnector.h"
#include "ProjectConstants.h"

#include <boost/lexical_cast.hpp>

using namespace std;

FileRecord::FileRecord()
{
    Init();
}

void FileRecord::Init()
{
    dbcon = DBConnector::GetConnection();

    Filename = "";
    CurrentVersionHash = 0;
    CurrentVersion = 0;
    NumberOfVersions = 0;
    ModifiedTime = 0;

    bIsValid = false;
}

FileRecord::~FileRecord()
{
    if(dbcon != NULL)
    {
        //dbcon->close();
    }

    //delete dbcon;
    dbcon = NULL;
}

FileRecord::FileRecord(std::string filename)
{
    Init();
}
```

```
RetrieveFileRecordFromDB(filename);
}

bool FileRecord::CreateFile(string filename, string newComment)
{
    sql::Statement *stmt = dbcon->createStatement();
    bool bSuccess = true;

    if(stmt == NULL)
    {
        //failed to get a connection to the database
        bSuccess = false;
    }

    bIsValid = false;
    try
    {
        //create file record
        if(bSuccess)
        {
            //beginning of statement
            string sqlstatement = "insert into File(filename, curhash, curversion, numversions) values(";
            //filename
            sqlstatement += "\"" + filename + "\"" + ", ";
            //curhash
            sqlstatement += boost::lexical_cast<string>(CurrentVersionHash) + ", ";
            //curversion
            sqlstatement += boost::lexical_cast<string>(CurrentVersion) + ", ";
            //numversions
            sqlstatement += boost::lexical_cast<string>(NumberOfVersions);
            //end of statement
            sqlstatement += ");";

            log(sqlstatement);

            bSuccess = stmt->executeUpdate(sqlstatement);
            dbcon->commit();
        }

        //retrieve record from DB
        if(bSuccess)
        {
            log("Retrieving record from db");
            RetrieveFileRecordFromDB(filename);
            bSuccess = IsValid();
        }

        if(bSuccess)
        {
            log("Adding new version");
            bSuccess = AddNewVersion(Filename, newComment);
        }
    }
}
```

```
}  
}  
catch (sql::SQLException &e)  
{  
log("ERROR: ");  
log(e.what());  
log(e.getErrorCode());  
log(e.getSQLState());  
bSuccess = false;  
}  
  
delete stmt;  
return bSuccess;  
}  
  
unsigned int FileRecord::GetCurrentVersionNumber()  
{  
return CurrentVersion;  
}  
  
bool FileRecord::UpdateRecordInDB()  
{  
sql::Statement *stmt = dbcon->createStatement();  
bool bSuccess = true;  
  
if(stmt == NULL)  
{  
//failed to get a connection to the database  
bSuccess = false;  
}  
  
if(IsValid() == false)  
{  
bSuccess = false;  
}  
  
try  
{  
//create file record  
if(bSuccess)  
{  
//beginning of statement  
string sqlstatement = "update File set ";  
//curhash  
sqlstatement += "curhash = " + boost::lexical_cast<string>(CurrentVersionHash) + ", ";  
//curversion  
sqlstatement += "curversion = " + boost::lexical_cast<string>(CurrentVersion) + ", ";  
//numversions  
sqlstatement += "numversions = " + boost::lexical_cast<string>(NumberOfVersions);  
//end of statement  
sqlstatement += " where filename = \"" + Filename + "\"";  
  
log(sqlstatement);  

```

```
bSuccess = stmt->executeUpdate(sqlstatement);
}
}
catch (sql::SQLException &e)
{
log("ERROR: ");
log(e.what());
log(e.getErrorCode());
log(e.getSQLState());
bSuccess = false;
}

delete stmt;
return bSuccess;
}

VersionRecord FileRecord::GetVersion(unsigned int versionNum)
{
VersionRecord newVersion(Filename, versionNum);
return newVersion;
}

vector<VersionRecord> FileRecord::GetAllVersions()
{
vector<VersionRecord> allVersions;

for(unsigned int i = 0; i < NumberOfVersions; i++)
{
VersionRecord newVersion = GetVersion(i + 1);

if(newVersion.IsValid())
{
allVersions.push_back(newVersion);
}
}

return allVersions;
}

void FileRecord::PurgeOldVersions(int numberOfVersionsToKeep)
{
for(unsigned int i = 0; i <= NumberOfVersions - numberOfVersionsToKeep; i++)
{
VersionRecord purgeVersion = GetVersion(i);
if(purgeVersion.IsValid())
{
log("Actual Purge Record No: " + boost::lexical_cast<string>(i));
purgeVersion.PurgeVersion();
}
}
}
```

```
int FileRecord::GetNumberOfVersions()
{
return NumberOfVersions;
}

//Ensures there is a valid corresponding record in the database
bool FileRecord::IsValid()
{
return bIsValid;
}

bool FileRecord::GetVersionFileContents(unsigned int requestedVersionNumber, string fileOutPath)
{
VersionRecord requestedVersion = GetVersion(requestedVersionNumber);
if(requestedVersion.IsValid())
{
return requestedVersion.GetFileData(fileOutPath);
}
return false;
}

std::string FileRecord::GetFilename()
{
return Filename;
}

unsigned int FileRecord::GetVersionSize(unsigned int versionNumber)
{
VersionRecord version = GetVersion(versionNumber);
if(version.IsValid())
{
return version.GetSize();
}
else
{
log("Invalid version, cannot retrieve version size");
return 0;
}
}

bool FileRecord::AddNewVersion(string NewFileVersionPath, string newComment)
{
bool bSuccess = true;
unsigned int newHash = 0;

log("Attempting to add new version from file: " + NewFileVersionPath);

if(boost::filesystem::exists(NewFileVersionPath) == false)
{
log("ERROR: File does not exist");
bSuccess = false;
}
}
```



```
if(bSuccess)
{
MurmurHash3_x86_32_FromFile(NewFileVersionPath, MURMUR_SEED_1, &newHash);

log("Hash generated for new version = " + boost::lexical_cast<string>(newHash));

//fail if hash matches existing
if(NumberOfVersions > 0 && CurrentVersionHash == newHash)
{
log("New version hash is no different. File is unchanged");
bSuccess = false;
}
}

//Add new version
VersionRecord newVersion;
if(bSuccess)
{
log("Adding new version");
bSuccess = newVersion.CreateVersion(Filename, NewFileVersionPath, CurrentVersion + 1, newHash,
newComment);
}

//Update version details
if(bSuccess)
{
log("New version added");
NumberOfVersions += 1;
CurrentVersion = newVersion.GetVersionNumber();
CurrentVersionHash = newVersion.GetHash();
bSuccess = UpdateRecordInDB();
}

return bSuccess;
}

bool FileRecord::IsChanged()
{
if(IsValid() == false)
{
return false;
}

if(boost::filesystem::exists(Filename) == false)
{
return false;
}

unsigned int fileHash;
MurmurHash3_x86_32_FromFile(Filename, MURMUR_SEED_1, &fileHash);

if(fileHash == CurrentVersionHash)
```

```
{
return false;
}
return true;
}

unsigned int FileRecord::GetHashOfFileBuffer(int FileLength, const char* FileBuffer)
{
uint32_t out;
MurmurHash3_x86_32(FileBuffer, FileLength, 10000, &out);
return out;
}

bool FileRecord::RetrieveFileRecordFromDB(string inFilename)
{
bIsValid = false;
try
{
// Run Query
sql::Statement *stmt = dbcon->createStatement();
sql::ResultSet *rs = stmt->executeQuery("select * from File where filename = '" + inFilename +
"'");

// Output Results
while(rs->next())
{
//count = rs->getUInt(1);
Filename = rs->getString("filename");
CurrentVersionHash = rs->getUInt("curhash");
CurrentVersion = rs->getUInt("curversion");
NumberOfVersions = rs->getUInt("numversions");
bIsValid = true;
}

delete rs;
rs = NULL;
delete stmt;
stmt = NULL;
}
catch (sql::SQLException &e)
{
log("ERROR: ");
log(e.what());
log(e.getErrorCode());
log(e.getSQLState());
}

return bIsValid;
}
```

```
/*
 * File:    GetCommentForm.h
 * Author:  io447
 *
 * Created on 21 September 2015, 4:35 PM
 */

#ifndef _GETCOMMENTFORM_H
#define _GETCOMMENTFORM_H

#include "ui_GetCommentForm.h"

class GetCommentForm : public QDialog{
    Q_OBJECT
public:
    GetCommentForm();
    virtual ~GetCommentForm();
public slots:
    void SetComment();
    QString GetComment();
private:
    Ui::GetCommentForm widget;
    QString comment;
};

#endif/* _GETCOMMENTFORM_H */
```

```
/*
 * File:    GetCommentForm.cpp
 * Author:  io447
 *
 * Created on 21 September 2015, 4:35 PM
 */
#include <QFileDialog>
#include <QApplication>
#include <iostream>
#include <string>

#include "GetCommentForm.h"

GetCommentForm::GetCommentForm() {
    widget.setupUi(this);

    connect(widget.buttonBoxComment, SIGNAL(accepted()), this, SLOT(SetComment()));
    connect(widget.buttonBoxComment, SIGNAL(rejected()), this, SLOT(reject()));
}

GetCommentForm::~GetCommentForm() {}

void GetCommentForm::SetComment()
{
    comment = widget.textGetCommentForm->toPlainText();
}

QString GetCommentForm::GetComment()
{
    return comment;
}
```

```
//-----  
// MurmurHash3 was written by Austin Appleby, and is placed in the public  
// domain. The author hereby disclaims copyright to this source code.  
  
#ifndef _MURMURHASH3_H_  
#define _MURMURHASH3_H_  
  
#include <string>  
  
//-----  
// Platform-specific functions and macros  
  
// Microsoft Visual Studio  
  
#if defined(_MSC_VER) && (_MSC_VER < 1600)  
  
typedef unsigned char uint8_t;  
typedef unsigned int uint32_t;  
typedef unsigned    int64 uint64_t;  
  
// Other compilers  
  
#else    // defined(_MSC_VER)  
  
#include <stdint.h>  
  
#endif // !defined(_MSC_VER)  
  
//-----  
  
void MurmurHash3_x86_32 ( const void * key, int len, uint32_t seed, void * out );  
  
void MurmurHash3_x86_32_FromFile( std::string filepath, uint32_t seed, void * out );  
  
void MurmurHash3_x86_128 ( const void * key, int len, uint32_t seed, void * out );  
  
void MurmurHash3_x64_128 ( const void * key, int len, uint32_t seed, void * out );  
  
//-----  
  
#endif // _MURMURHASH3_H_
```

```
//-----  
// MurmurHash3 was written by Austin Appleby, and is placed in the public  
// domain. The author hereby disclaims copyright to this source code.  
  
// Note - The x86 and x64 versions do _not_ produce the same results, as the  
// algorithms are optimized for their respective platforms. You can still  
// compile and run any of them on any platform, but your performance with the  
// non-native version will be less than optimal.  
  
#include "MurmurHash3.h"  
  
#include <string>  
#include <fstream>  
using namespace std;  
  
//-----  
// Platform-specific functions and macros  
  
// Microsoft Visual Studio  
  
#if defined(_MSC_VER)  
  
#define FORCE_INLINE    forceinline  
  
#include <stdlib.h>  
  
#define ROTL32(x,y)    _rotl(x,y)  
#define ROTL64(x,y)    _rotl64(x,y)  
  
#define BIG_CONSTANT(x) (x)  
  
// Other compilers  
  
#else    // defined(_MSC_VER)  
  
#define FORCE_INLINE inline attribute ((always_inline))  
  
inline uint32_t rotl32 ( uint32_t x, int8_t r )  
{  
    return (x << r) | (x >> (32 - r));  
}  
  
inline uint64_t rotl64 ( uint64_t x, int8_t r )  
{  
    return (x << r) | (x >> (64 - r));  
}  
  
#define ROTL32(x,y)    rotl32(x,y)  
#define ROTL64(x,y)    rotl64(x,y)  
  
#define BIG_CONSTANT(x) (x##LLU)  
  
#endif // !defined(_MSC_VER)
```

```
//-----  
// Block read - if your platform needs to do endian-swapping or can only  
// handle aligned reads, do the conversion here  
  
FORCE_INLINE uint32_t getblock32 ( const uint32_t * p, int i )  
{  
    return p[i];  
}  
  
FORCE_INLINE uint64_t getblock64 ( const uint64_t * p, int i )  
{  
    return p[i];  
}  
  
//-----  
// Finalization mix - force all bits of a hash block to avalanche  
  
FORCE_INLINE uint32_t fmix32 ( uint32_t h )  
{  
    h ^= h >> 16;  
    h *= 0x85ebca6b;  
    h ^= h >> 13;  
    h *= 0xc2b2ae35;  
    h ^= h >> 16;  
  
    return h;  
}  
  
//-----  
  
FORCE_INLINE uint64_t fmix64 ( uint64_t k )  
{  
    k ^= k >> 33;  
    k *= BIG_CONSTANT(0xff51afd7ed558ccd);  
    k ^= k >> 33;  
    k *= BIG_CONSTANT(0xc4ceb9fe1a85ec53);  
    k ^= k >> 33;  
  
    return k;  
}  
  
//-----  
  
void MurmurHash3_x86_32 ( const void * key, int len,  
uint32_t seed, void * out )  
{  
    const uint8_t * data = (const uint8_t*)key;  
    const int nblocks = len / 4;  
  
    uint32_t h1 = seed;  
  
    const uint32_t c1 = 0xcc9e2d51;
```

```
const uint32_t c2 = 0x1b873593;

//-----
// body

const uint32_t * blocks = (const uint32_t *) (data + nblocks*4);

for(int i = -nblocks; i; i++)
{
    uint32_t k1 = getblock32(blocks,i);

    k1 *= c1;
    k1 = ROTL32(k1,15);
    k1 *= c2;

    h1 ^= k1;
    h1 = ROTL32(h1,13);
    h1 = h1*5+0xe6546b64;
}

//-----
// tail

const uint8_t * tail = (const uint8_t*) (data + nblocks*4);

uint32_t k1 = 0;

switch(len & 3)
{
case 3: k1 ^= tail[2] << 16;
case 2: k1 ^= tail[1] << 8;
case 1: k1 ^= tail[0];
    k1 *= c1; k1 = ROTL32(k1,15); k1 *= c2; h1 ^= k1;
};

//-----
// finalization

h1 ^= len;

h1 = fmix32(h1);

*(uint32_t*)out = h1;
}

void MurmurHash3_x86_32_FromFile( string filepath, uint32_t seed, void * out )
{
    int len = 0;
    ifstream ins(filepath.c_str());

    if(ins.is_open() == false)
    {
        //could not open the file
    }
}
```



```

return;
}

//get file length
ins.seekg(0, ios::end);
len = ins.tellg();
ins.seekg(0, ios::beg);

//const uint8_t * data = (const uint8_t*)key;
const int nblocks = len / 4;

uint32_t h1 = seed;

const uint32_t c1 = 0xcc9e2d51;
const uint32_t c2 = 0x1b873593;

//-----
// body

//const uint32_t * blocks = (const uint32_t *) (data + nblocks*4);

for(int i = -nblocks; i; i++)
{
uint32_t k1;
//ins >> k1;
ins.get((char*)&k1, 4);
//ins.read(&k1, 4);

k1 *= c1;
k1 = ROTL32(k1,15);
k1 *= c2;

h1 ^= k1;
h1 = ROTL32(h1,13);
h1 = h1*5+0xe6546b64;
}

//-----
// tail

const int tailLength = len - (nblocks * 4);
uint8_t* tail = new uint8_t[tailLength];
for( int i = 0; i < tailLength; i++)
{
tail[i] = ins.get();
}

uint32_t k1 = 0;

switch(len & 3)
{
case 3: k1 ^= tail[2] << 16;
case 2: k1 ^= tail[1] << 8;

```

```

case 1: k1 ^= tail[0];
k1 *= c1; k1 = ROTL32(k1,15); k1 *= c2; h1 ^= k1;
};

//-----
// finalization

h1 ^= len;

h1 = fmix32(h1);

*(uint32_t*)out = h1;
}

//-----

void MurmurHash3_x86_128 ( const void * key, const int len,
uint32_t seed, void * out )
{
const uint8_t * data = (const uint8_t*)key;
const int nblocks = len / 16;

uint32_t h1 = seed;
uint32_t h2 = seed;
uint32_t h3 = seed;
uint32_t h4 = seed;

const uint32_t c1 = 0x239b961b;
const uint32_t c2 = 0xab0e9789;
const uint32_t c3 = 0x38b34ae5;
const uint32_t c4 = 0xale38b93;

//-----
// body

const uint32_t * blocks = (const uint32_t *) (data + nblocks*16);

for(int i = -nblocks; i; i++)
{
uint32_t k1 = getblock32(blocks,i*4+0);
uint32_t k2 = getblock32(blocks,i*4+1);
uint32_t k3 = getblock32(blocks,i*4+2);
uint32_t k4 = getblock32(blocks,i*4+3);

k1 *= c1; k1 = ROTL32(k1,15); k1 *= c2; h1 ^= k1;

h1 = ROTL32(h1,19); h1 += h2; h1 = h1*5+0x561ccd1b;

k2 *= c2; k2 = ROTL32(k2,16); k2 *= c3; h2 ^= k2;

h2 = ROTL32(h2,17); h2 += h3; h2 = h2*5+0x0bcaa747;

k3 *= c3; k3 = ROTL32(k3,17); k3 *= c4; h3 ^= k3;

```

```
h3 = ROTL32(h3,15); h3 += h4; h3 = h3*5+0x96cd1c35;

k4 *= c4; k4 = ROTL32(k4,18); k4 *= c1; h4 ^= k4;

h4 = ROTL32(h4,13); h4 += h1; h4 = h4*5+0x32ac3b17;
}

//-----
// tail

const uint8_t * tail = (const uint8_t*)(data + nbblocks*16);

uint32_t k1 = 0;
uint32_t k2 = 0;
uint32_t k3 = 0;
uint32_t k4 = 0;

switch(len & 15)
{
case 15: k4 ^= tail[14] << 16;
case 14: k4 ^= tail[13] << 8;
case 13: k4 ^= tail[12] << 0;
    k4 *= c4; k4 = ROTL32(k4,18); k4 *= c1; h4 ^= k4;

case 12: k3 ^= tail[11] << 24;
case 11: k3 ^= tail[10] << 16;
case 10: k3 ^= tail[ 9] << 8;
case  9: k3 ^= tail[ 8] << 0;
    k3 *= c3; k3 = ROTL32(k3,17); k3 *= c4; h3 ^= k3;

case  8: k2 ^= tail[ 7] << 24;
case  7: k2 ^= tail[ 6] << 16;
case  6: k2 ^= tail[ 5] << 8;
case  5: k2 ^= tail[ 4] << 0;
    k2 *= c2; k2 = ROTL32(k2,16); k2 *= c3; h2 ^= k2;

case  4: k1 ^= tail[ 3] << 24;
case  3: k1 ^= tail[ 2] << 16;
case  2: k1 ^= tail[ 1] << 8;
case  1: k1 ^= tail[ 0] << 0;
    k1 *= c1; k1 = ROTL32(k1,15); k1 *= c2; h1 ^= k1;
};

//-----
// finalization

h1 ^= len; h2 ^= len; h3 ^= len; h4 ^= len;

h1 += h2; h1 += h3; h1 += h4;
h2 += h1; h3 += h1; h4 += h1;

h1 = fmix32(h1);
```

```
h2 = fmix32(h2);
h3 = fmix32(h3);
h4 = fmix32(h4);

h1 += h2; h1 += h3; h1 += h4;
h2 += h1; h3 += h1; h4 += h1;

((uint32_t*)out)[0] = h1;
((uint32_t*)out)[1] = h2;
((uint32_t*)out)[2] = h3;
((uint32_t*)out)[3] = h4;
}

//-----

void MurmurHash3_x64_128 ( const void * key, const int len,
const uint32_t seed, void * out )
{
const uint8_t * data = (const uint8_t*)key;
const int nblocks = len / 16;

uint64_t h1 = seed;
uint64_t h2 = seed;

const uint64_t c1 = BIG_CONSTANT(0x87c37b91114253d5);
const uint64_t c2 = BIG_CONSTANT(0x4cf5ad432745937f);

//-----
// body

const uint64_t * blocks = (const uint64_t *) (data);

for(int i = 0; i < nblocks; i++)
{
uint64_t k1 = getblock64(blocks,i*2+0);
uint64_t k2 = getblock64(blocks,i*2+1);

k1 *= c1; k1 = ROTL64(k1,31); k1 *= c2; h1 ^= k1;

h1 = ROTL64(h1,27); h1 += h2; h1 = h1*5+0x52dce729;

k2 *= c2; k2 = ROTL64(k2,33); k2 *= c1; h2 ^= k2;

h2 = ROTL64(h2,31); h2 += h1; h2 = h2*5+0x38495ab5;
}

//-----
// tail

const uint8_t * tail = (const uint8_t*) (data + nblocks*16);

uint64_t k1 = 0;
uint64_t k2 = 0;
```

```
switch(len & 15)
{
case 15: k2 ^= ((uint64_t)tail[14]) << 48;
case 14: k2 ^= ((uint64_t)tail[13]) << 40;
case 13: k2 ^= ((uint64_t)tail[12]) << 32;
case 12: k2 ^= ((uint64_t)tail[11]) << 24;
case 11: k2 ^= ((uint64_t)tail[10]) << 16;
case 10: k2 ^= ((uint64_t)tail[ 9]) << 8;
case  9: k2 ^= ((uint64_t)tail[ 8]) << 0;
    k2 *= c2; k2 = ROTL64(k2,33); k2 *= c1; h2 ^= k2;

case  8: k1 ^= ((uint64_t)tail[ 7]) << 56;
case  7: k1 ^= ((uint64_t)tail[ 6]) << 48;
case  6: k1 ^= ((uint64_t)tail[ 5]) << 40;
case  5: k1 ^= ((uint64_t)tail[ 4]) << 32;
case  4: k1 ^= ((uint64_t)tail[ 3]) << 24;
case  3: k1 ^= ((uint64_t)tail[ 2]) << 16;
case  2: k1 ^= ((uint64_t)tail[ 1]) << 8;
case  1: k1 ^= ((uint64_t)tail[ 0]) << 0;
    k1 *= c1; k1 = ROTL64(k1,31); k1 *= c2; h1 ^= k1;
};

//-----
// finalization

h1 ^= len; h2 ^= len;

h1 += h2;
h2 += h1;

h1 = fmix64(h1);
h2 = fmix64(h2);

h1 += h2;
h2 += h1;

((uint64_t*)out)[0] = h1;
((uint64_t*)out)[1] = h2;
}

//-----
```

```
/*
 * File:    MyWindow.h
 * Author:  io447
 *
 * Created on 21 September 2015, 12:36 PM
 */

#ifndef _MYWINDOW_H
#define _MYWINDOW_H

#include "ui_MyWindow.h"
#include "GetCommentForm.h"
#include "RetrieveForm.h"
using namespace std;

class FileArchiver;
class FileRecord;
typedef FileArchiver* FilePtr;
class TableModel;

class MyWindow : public QMainWindow {
    Q_OBJECT
public:
    MyWindow();
    virtual ~MyWindow();

public slots:
    //select file
    void SelectFile();
    void SaveCurrent();
    void ShowComment();
    void CreateFirstVersion(std::string fileName);
    void AddNewVersion(std::string fileName);
    void RetrieveVersionDataForFile();
    void RetrieveVersion();
    void SetReferenceVersion();

private:
    Ui::MyWindow widget;
    GetCommentForm * getCommentWindow;
    RetrieveForm * retrieveWindow;
    TableModel * tablemodel;
    QString fileName;
};

#endif/* _MYWINDOW_H */
```