Minecraft Launcher

Comp4 Project

Contents

Analysis

Background and Identification of Problem

Minecraft is commonly used at schools for educational purposes. Often, there is a server and client component, to allow students to explore a “world” together, and encourage constructive teamwork. This project aims simplify the complexity behind maintaining multiple clients and a server at school.

The server may include “mods” that affect the gameplay, for example, by adding new tools, physics and permissions. The client must include the same mods as the server, in order to be able to connect to it. The client must be running the same Minecraft version as the server, and the mods must be compatible with the version of Minecraft. Failing this, Minecraft will fail to launch, or connect to the server.

Normally, a user must ensure they are running the same version of Minecraft. They then must acquire the mods, usually by downloading them off the internet. The amount of mods may vary between 1 and 100, depending on the server. This can make the task non-trivial for a standard user, however in a school, client installations should be identical.

This process can lead to mistakes and crashes, and hence the tedious and time consuming cycle of debugging must occur on the offending computers, which must then be again updated. Should the mods change, all clients must then be updated.

Therefore, this project aims to provide a Minecraft Launcher that will be able to produce and synchronise a working copy of Minecraft, along with the server operator’s customised configuration files and mods, similar to a version control system. It should provide user management facilities, through the use of a relational database, which will store session data about users such as the log on time, log off time, and computer name, whilst also storing a history of messages sent in Minecraft. This will allow the teacher to maintain a safe user environment, and review incidents without wasting time searching log files.

The few systems that exist currently re-download all files, however it is far quicker and more resourceful to only download the changes. This presents another problem of recognising change, such as when a mod file with the same name has been updated, however the contents of the mod are different.

This will allow the server operator to themselves produce a working copy of Minecraft, and then synchronize it, ensuring all clients are up to date with a tested copy, similar to a version control system.

Description of Current System

When the student launches Minecraft, they are presented briefly with the following screen:

This is a launcher written by the teacher, Mr Macleod, in an attempt to manage the distribution of client computers running Minecraft. The program itself is a small Delphi application that looks for a list of commands under "'K:\COMPUTING\MINECRAFTEDU\UPDATES\update.txt". A sample of the update.txt file is below:

[message] Updating Mod Files ... please wait
[copy] minecraftedusettings.ini
[message] Reading the update data file ....
[delete] \mods\1.7.10\ComputerCraft1.65.jar
[copy] \mods\1.7.10\ComputerCraft+ComputerCraftEdu1.65.jar
[copy] \mods\1.7.10\ArchimedesShips-1.7.1.jar
[delete] \mods\1.7.10\SecurityCraft v1.6.1 for 1.7.10.jar
[delete] \mods\1.7.10\brainstonemod.zip
[copy] \mods\1.7.10\customnpcs_1.7.10b.jar
[delete] \mods\1.6.4\customnpcs_1.6.4.zip
[delete] \settings\allowedstudentcommands.txt
[delete] \mods\1.7.10\ComputerCraft-Mod-1.7.10.jar
[message] Complete, thank you for waiting.

The program identifies the command, one of [message], [delete], [copy], and carries out the relevant and self-explanatory action. Mr Macleod must type out each line and ensure that there are no errors in the file path, the command, and the spacing must be constant.

A table of aliases are read from the "K:\COMPUTING\MINECRAFTEDU\MCALIAS.TXT" file. The program checks if the user logged into Windows in listed in the MCALIAS file and changes their name in Minecraft to that of the alias. It also contains the server address, and any banned players

A sample of the file MCALIAS is below:

server,10.3.222.90
gnm,MrMacleod
delphi_h_singh,Harjot
delphi_t_nicholls,Tim
delphi_v_chhapwale,Herobrine
###################### banned
12ipatel,banned
12kbamrah,banned

Mr Macleod also must type this list out manually, in a fixed format.

This program then launches a MinecraftEdu bootstrap launcher, which executes the Java command required to launch Minecraft with configuration and mods. This is hidden to the user, and diagnosing any failures at this stage is difficult.

Overall, the system is extremely error prone, as a single mistake in the MCALIAS or UPDATE.TXT file will cause a crash (this includes spacing errors). The system is inflexible, as it only offers the 3 commands described above, which only allow for file manipulation.

Identification of Prospective User(s)

The users in this case are fairly obvious:

1. Mr Macleod, who maintains and runs the server. Often he is required to manage a deployment across 2-3 IT labs; at least 60 computers. This can prove troublesome to manage without an automated and bespoke solution tuned to the school's infrastructure requirements. If he wishes to install a new mod on the server, all computers must be updated. With this many computers, an installation can easily go wrong using a manual process. Thus it makes sense to use an application that will mirror a working copy exactly.

2. The students, who will be playing Minecraft across these computers. Each computer has 1 licenced copy of Minecraft, and any user should be able to play, identified by their user account. These students simply launch Minecraft and connect to the server to play. If an issue occurs, they must contact Mr Macleod personally and resolve an issues in this manner.

Identification of User Needs and Acceptable Limitations

Interview with Mr Macleod

Me: What is the biggest issue with the current launcher?

Mr Macleod: I have to write out commands, which conform to a very strict format. These commands simply manipulate files, and it is hard to find mistakes in these commands. If I am wrong, there is no indication of this. This makes me less keen to update Minecraft or install new mods.

Me: I have put together an initial list of objectives, based on what the old launcher does. The launcher should be able to launch different versions of Minecraft, synchronize files automatically, and provide configuration options for launching Minecraft. Do you wish to change any of these initial suggestions?

Mr Macleod: No, they are exactly what I would like the launcher to do.

Me: What, aside from already described, should the launcher be able to do?

Mr Macleod: The launcher should do more than simply launch Minecraft. In the school environment, I often have administrative issues, and incident management becomes difficult. For example, if a pupil mentions that they have received nasty messages from another user, it is extremely time consuming and tedious to collate log files, go through them, and sanction the pupil. Some sort of administrative panel that allows me to carry out these functions or search would make it much easier to ensure that the Minecraft experience is a safe and fair one, especially for the younger pupils. It may be a good idea to include some form of feedback facility.

Me: What would this entail?

Mr Macleod: The ability to request a new mod may prevent students unnecessarily emailing me with these requests. Additionally, the administrator, me, should only be able to authorise name changes, so perhaps a facility to automate the procedure of requesting a name and me approving it could be implemented?

Me: Are pupils required to adhere to any time policies?

Mr Macleod: No, although actually only permitting the pupil to play for x minutes each day may help with the congestion in IT labs, and allow it for other use.

Me: Would you be willing to run a HTTP server/MySQL database server on the Minecraft server machine?

Mr Macleod: Yes, that shouldn't be a problem. I will not know how to set it up though, so please include this in the user guide.

Me: Do you require the launcher for other platforms?

Mr Macleod: No, although we may acquire some Macs in the future, so the source code may need to be at least ready to modify for this.

Me: What kind of design do you have in mind?

Mr Macleod: The design should be user friendly, but also very minimalistic. The user should be aware that they simply need to press a button to play Minecraft. The launcher should be aesthetically pleasing, and should give a sense of relaxation.

Me: How would you like to see the launcher carry out file management?

Mr Macleod: There should definitely be no effort required on the student side, as this may cause issues. Ideally, their launchers should download the correct files when Minecraft is launched. I think that the process for updating files should be easy for me. Perhaps it could copy all the files of my directory into a public space. It may need to exclude certain files, though.

Me: Would you be willing to use a regular expression to determine which files should not be copied to the public space?

Mr Macleod: A regex seems like a very powerful way of accomplishing this task. I am familiar with regexes, and do think this would be ideal.

Me: Minecraft changes version fairly frequently. Do you wish to be able to change version of Minecraft from within the launcher?

Mr Macleod: That would be incredibly useful. In fact, it would be nice if the launcher could display a list of versions, and I could select the one I wished to install. In case installing a new version does not work out, I would like to be able to create a backup of the old Minecraft installation, and restore it whenever necessary.

Survey

50 sixth formers were surveyed informally, and asked about how Minecraft affected their lives. This survey was used to identify how the launcher could be designed to better suit a school's needs. Only 4 out of the 50 admitted to commonly playing Minecraft, while 38 out of 50 suggested that Minecraft has made the computing labs unusable during lunch and break times, due to masses of younger pupils taking up the computers for the entire break. Therefore, it was made very clear that pupils should have a form of time management applied, perhaps using a daily quota system, which would limit usage, and allow the rest of the school community to regain the computers for work. It also prevents younger pupils from skipping lessons to play Minecraft, and becoming addicted.

Personal Experience

I manage a Minecraft server for a small, but specific community of people, and face similar challenges in terms of ensuring all members are able to play Minecraft in a hassle-free manner. Members/users are required to download the same mods as the server has, and this needs to be updated. My users only play on my server, which is similar to the situation at a school, and thus understand the importance of seamless mod syncing/file updates, and the hassle of dealing with individual user cases, which is why a file-repository to clone files from will be suggested in the objectives.

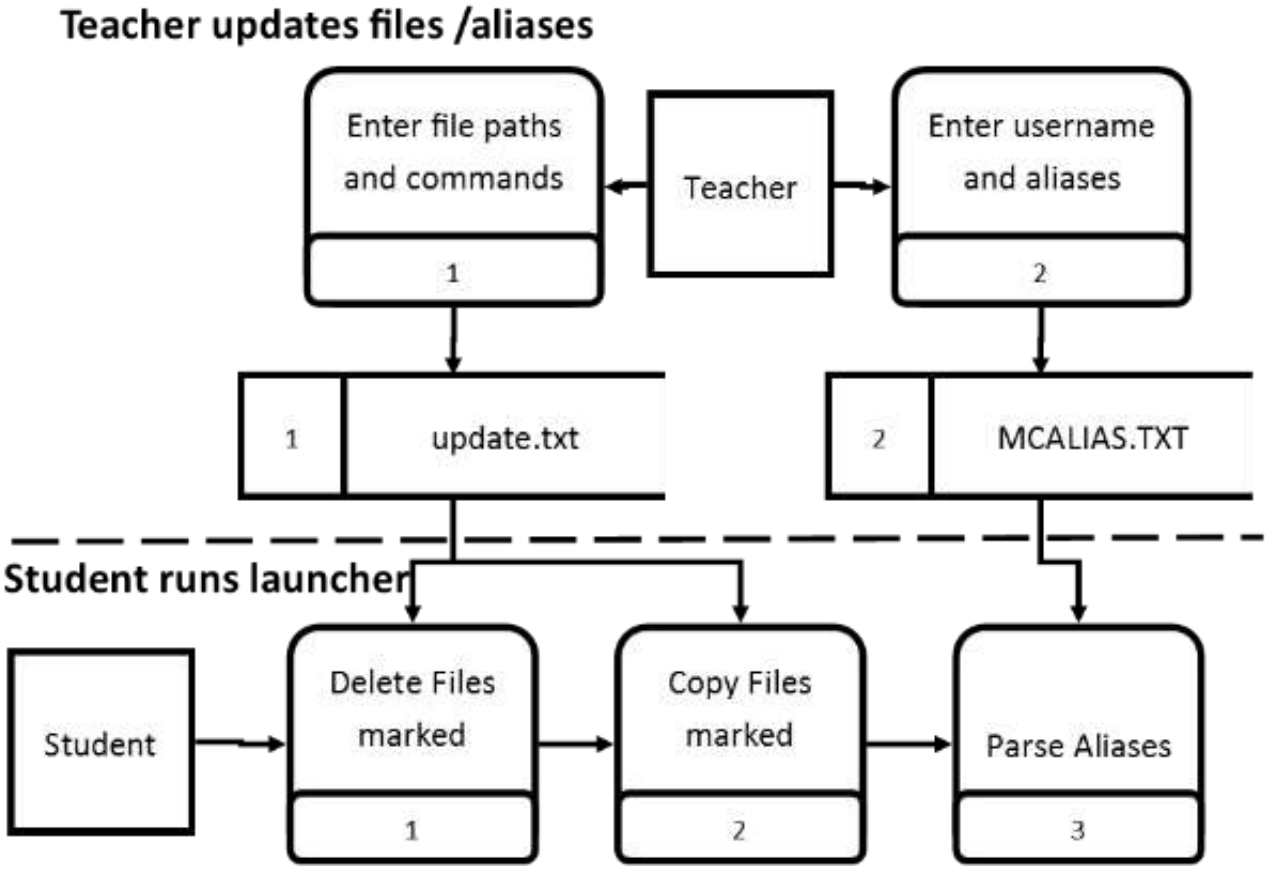
Conclusion

My meeting with Mr Macleod raised an area of concern, regarding the administration of user activity, which is obviously very important in a school environment. The survey gave some insight into how Minecraft affected the school as a whole, and how the launcher could be used to provide some form of control, and restore some balance to computer usage. This resulted in the following:

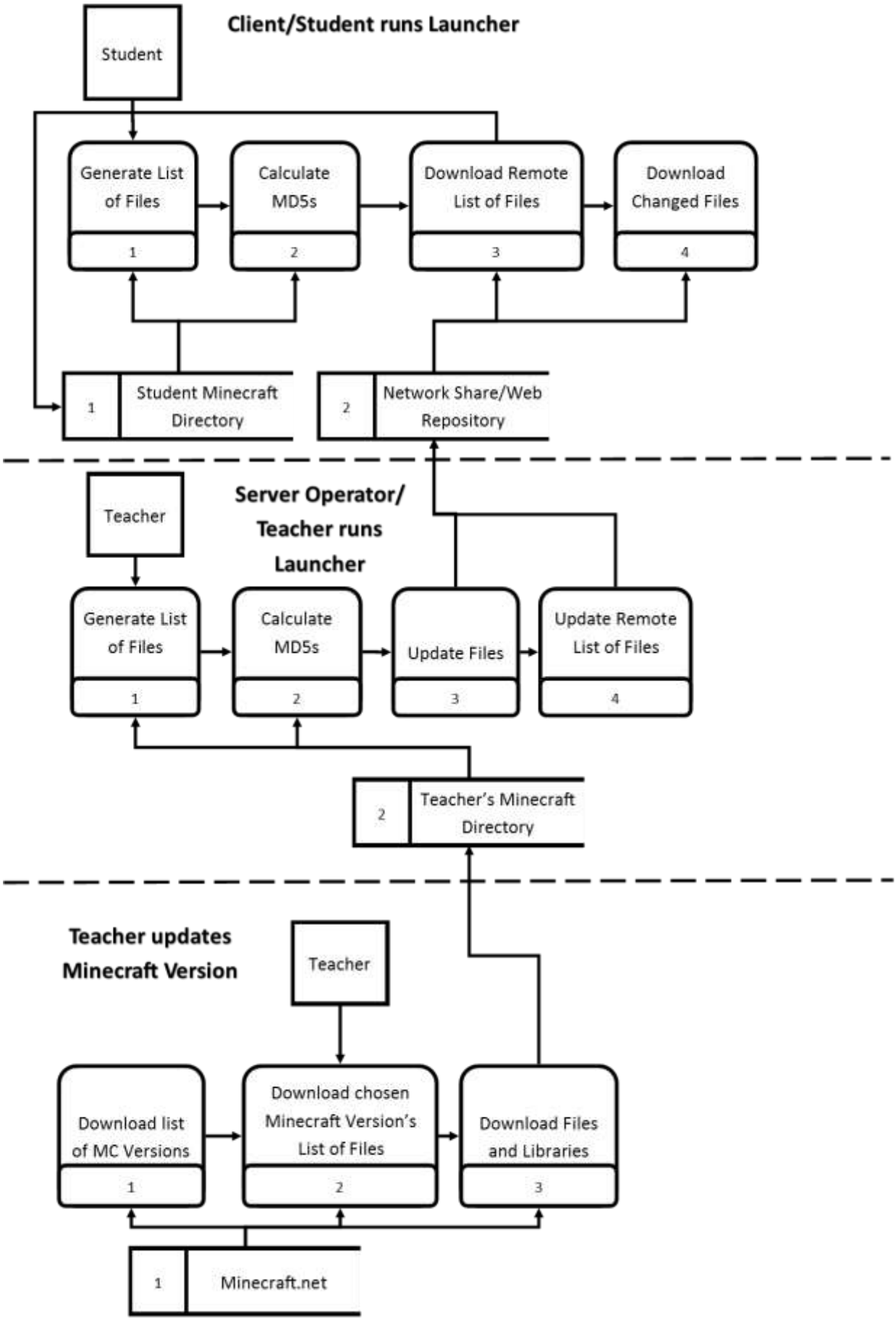
- Confirmed initially planned objectives and added new objectives
    - Store log off and log on times to help provide evidence for student conflict
    - Set time quota, such that the user is only allowed to play for a certain time period
- Confirmed that a facility for mod requests/alias requests is required
- Gave permission for the setup of a MySQL database, residing on the Minecraft server, which could now be used to store and retrieve:
    - Aliases
    - Log on/Log off Times
    - Messages sent
    - Time Quotas
    - Mod requests
- An apache HTTP server, which could:
    - Store the current repository
    - Serve mod files to clients
- A control panel, or text file will be required to set and administrate student settings
- Use a multiplatform language/framework, considering the future

Data Flow Diagram

Existing



Proposed


Client/Student runs Launcher
Student
Generate List of Files
1
Calculate MD5s
2
Download Remote List of Files
3
Download Changed Files
4
1
Student Minecraft Directory
2
Network Share/Web Repository
Server Operator/ Teacher runs Launcher
Teacher
Generate List of Files
1
Calculate MD5s
2
Update Files
3
Update Remote List of Files
4
2
Teacher's Minecraft Directory
Teacher updates Minecraft Version
Teacher
Download list of MC Versions
1
Download chosen Minecraft Version's List of Files
2
Download Files and Libraries
3
1
Minecraft.net

Objectives of the Proposed System

1. The system must be able to launch Minecraft, using the Java Runtime Environment
2. The system must identify changes between the repository's version of Minecraft and the student's copy of Minecraft, when the student runs the system
3. The system must then download any changes; ideally in a pooled, multithreaded format, and transform the Minecraft directory such that it reflects the repository's state
4. The system should recognise the teacher is logged in and present different options accordingly
5. The system should store a log of events that occur, locally in the Minecraft directory
6. The system should allow the teacher to copy their version of Minecraft to the repository as the working copy, being able to exclude files with a regex
7. The teacher should be able to upgrade/downgrade available Minecraft versions within the system
8. The system should be able to handle aliases between a school user account and Minecraft name
9. The system should have an alias request feature or mod request feature which notifies the teacher
10. The teacher should be able to modify the parameters for launching the Java Virtual Machine
11. Log off and log on times should be stored and retrievable
12. The teacher should be able to set a time quota that students are able to play Minecraft for
13. The system should log all messages sent by the user so the teacher can review incidents
14. The teacher should be able to ban students
15. Messages, users and session information should all be searchable
16. The system should present some statistics to the teacher, such as the average time per user
17. The system should be able to save backups of a Minecraft installation, and restore them

Data Sources and Destinations

There are several data sources for the system. Primarily, the teacher's Minecraft folder/repository, where he will add mods and mirror a working copy of Minecraft.

The second data source is the MySQL database that will be used to store details about users, the messages they send, and session data.

The third data source is the online Minecraft.net servers, which serves data explaining which parameters to use when launcher Minecraft. Additionally, every time a user opens the launcher, a connection to the server database is made, which updates activity logs.

The data destination will be the student's Minecraft directory. This is unknown to the user. The program will also aggregate and display user information to the teacher from the database.

Data Volumes

The size of the files ranges from ~30 megabytes to a few hundred megabytes, depending on the number of mods the server operator has chosen to install. However not all the data needs to be downloaded every time, unless all the files have changed (a plausible case when a major upgrade occurs). Thus each time the launcher is opened, kilobytes worth of data in the form of text lists are downloaded to determine whether additional downloading is required. Only list data will reside in memory, and actual mod files will be saved directly to the hard disk.

Data Dictionary

Term	Definition
Minecraft	A game that allows the user to roam freely, build, and live in a virtual world of blocks. The game can be played in multiplayer mode, via a Minecraft server, and allows collaboration between various players. The game is written in Java, and launched using the Java Runtime Environment (JRE)
Minecraft Version	There are many different versions of the game, as it is updated frequently with new mods and features. The version number is in the format x.y.z, where a difference in y generally suggests a major change.
Minecraft Server	A running instance of the Minecraft server software, provided by Mojang. Usually run on a separate computer, with dedicated resources.
Minecraft Client	A running instance of the actual Minecraft game, which connects to the server and sends/receives updates to/from it. It contains a graphical user interface (GUI) to play Minecraft with.
Mod	A file that modifies the default behaviour of Minecraft in some way. An example could be a mod that adds better physics into the game, or in a school environment, a mod which gives the teacher some control in-game.
Minecraft Forge	To load these mods into Minecraft, there needs to be a bootstrapper/mod loader. Minecraft Forge is a mod loader that scans the "mod" directory when Minecraft is launched, and loads them into memory.
Session	A session consists of a log on and log off time, and the computer from which Minecraft was accessed. The session starts when Minecraft is launched, and ends when Minecraft is no longer running.
Quota	A daily time limit that a student is able to play Minecraft for. After this time has expired, the user should not be able to play Minecraft.
Message	Users are able to send messages in-game to other users. A message consists of the message text, and the time sent.
Banned	Users may cause trouble on servers, so an administrator may "ban" them from joining a server. This prevents them from playing Minecraft.
Alias	The in-game username of a player. This may be different to their Windows login name.
JRE 7	The Java Runtime Environment, version 7. This is a set of programs and utilities that allow Java applications to be executed. Minecraft requires JRE 7 or above, and can be downloaded freely online.
JVM	The Java Virtual Machine. All Java applications are executed and managed by the JVM. Memory management and garbage collection is also handled by the JVM, so it may be necessary to tune the parameters that the JVM runs Minecraft with, for better performance.
Apache	Apache is a HTTP webserver that is freely available online. It can be used to serve files, with minimal setup required.
MySQL	MySQL is a relational database technology that can be used to store persistent data with relations. Data can be accessed using SQL (server query language), and also presents the opportunity to manipulate data.
Minecraft Directory	The user-definable directory which contains all Minecraft-related files. By default, this is set to %appdata%/.minecraft, but can be changed.
Library	A Java library file that contains functions and methods that are used to run Minecraft. These are not platform specific, and found in the "libraries" folder, under the Minecraft directory.
Natives	A set of platform-dependant files that are required by Minecraft. An example of these may be OpenGL graphics libraries, which must be uniquely compliled for platforms. These are also usually architecture specific (32 or 64 bit).

Realistic Appraisal of the Feasibility of Potential Solutions

Modifying the Current System to Sync Mods with the Teacher's Minecraft Folder

The teacher's Minecraft folder could be copied into a shared folder, as it is now. The program would be modified to check which files need to be re-downloaded, perhaps by comparing the date modified or file size. This does not allow the teacher much control and no administrative capabilities. The main difference is that the teacher does not have to type out commands as before, they can instead be generated by an accompanying tool. This solution would be a step up for the teacher, however still does not provide any benefits to the students. The current solution does allow for the regulation of user names, session management, or message logging, and does not meet the proposed objectives. The current program is a console application, so cannot be modified to provide aesthetic visual feedback, nor any form of configuration or dynamic menus.

Pre-existing Minecraft Launchers

An example at http://www.atlauncher.com/, which is a customisable launcher that can be distributed and configured. It can store backups and switch between mod configurations. It is, however, primarily designed to aid users managing themselves, so the settings will have to be stored on a network drive. There are no facilities for handling or managing multiple users from an administration perspective. These launchers generally provide the ability to install and sync "mod packs", but this usually has to be initiated on each computer, and so relies on students using the launcher appropriately. The teacher could prepare a "mod pack", set up a working copy of the launcher, and distribute this to all other machines, however will be required to repeat this process if there are any significant changes to options. The students also have the ability to configure the launcher themselves, and this increases the chance of failure. Again, a custom launcher will not be able to log messages, store session data, or enforce quotas, as required by the proposed objectives. Issues may arise trying to use a custom launcher within the school network, as Minecraft.net and other sources are blocked for students. This may cause the launchers to crash, or be unable to retrieve data or "mod packs".

Bespoke Launcher with Administrative Control Panel

This launcher would be designed to work with similar set-ups, by using a local SQL database to store administrative and user specific details. Aside from syncing the mods, the launcher would be able to accommodate for students' mod requests/alias change requests. The very nature of a bespoke solution will mean the proposed objectives can be met in a manner ideal for the school environment.

Justification of Chosen Solution

The chosen bespoke solution allows Mr Macleod to do everything required by the specification in a unified and simple manner. Any errors are much easier to diagnose, considering that the launcher will store a log locally.

The launcher will be able to log user data, messages, store session data, and request data, using an SQL database.

The solution would also not encounter any problems regarding the network, since all students download files from a local server, and do not have to access restricted domains, such as Minecraft.net, which is blocked.

Graphical user interface (GUI) design can be carried out in such a way that it makes launching Minecraft a straightforward process for the students, while retaining the more advanced features for administrative use.

Comparison of Database Technologies

Initially Microsoft Access was looked at as the database technology. However, further research indicates that there are a number of problems with this:

- There is a maximum file size of an Access database
- Only accessible if the database file can be accessed. This presents security issues, as the database must be stored in an area with read and write access
- Access databases are relatively slow

Therefore, more appropriate technologies, MySQL and PostgreSQL, were investigated:

MySQL is a very common relational database technology, maintained by Oracle:

- Lightweight
- Can be accessed via a URL
- Tested and used by millions of heavy applications daily, without a fault
- Conforms to basic SQL standards
- Easy installation, part of the XAMPP software package

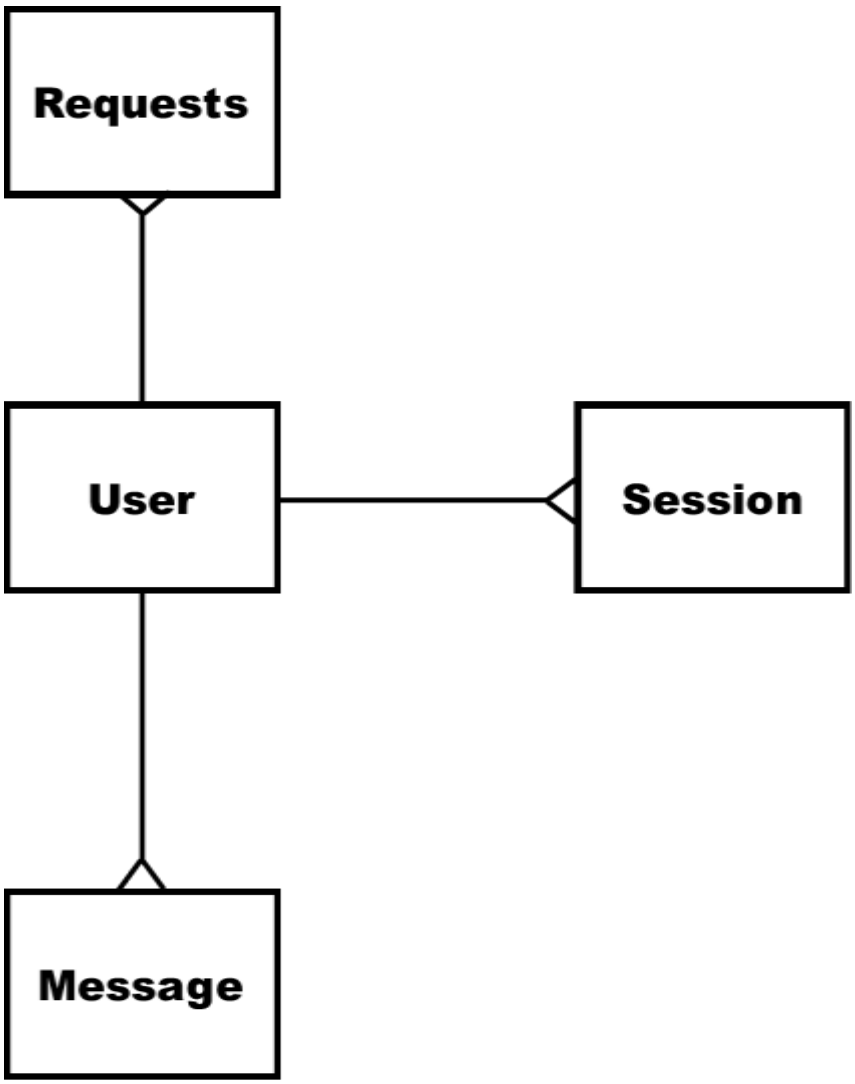
PostgreSQL is similar to MySQL, but has some fundamental differences:

- Very rich and powerful SQL functions
- Heavier/higher memory usage
- Similar query speeds to MySQL
- More complicated to setup

As a result of this comparison, MySQL was chosen, since only basic SQL was required, and it was more important that the user finds it easier to setup. Using the XAMPP package, the user is able to install the Apache HTTP server and MySQL in one installation, and is provided with a user friendly control panel for management. This is why the Apache HTTP server was chosen over Nginx or Lighttpd, since again, user installation was more critical than a minimal speed increase.

E-R Diagram

The database will be built, based on the simplified E-R diagram below:



The E-R diagram revolves around the user. The user is likely to have properties, such as name, alias, user id, and whether they are banned or not. There is a one-to-many relationship between the user and all other tables. This means one user can have messages, sessions, or requests. Messages are recorded so that the teacher can easily track who has said what, and when, and cross-reference this data with session data, of which user has logged on and off at what times. Requests are a convenience feature that allows a user to submit a request to the teacher, such as a alias/name change request, or a new mod request.

Generally, a user will have multiple messages within one session, although this is not a predetermined relationship. Using the session count and length, the teacher may be provided with statistics, which can be used to fine-tune the Minecraft experience to a particular year group that often play the game.

Due to the nature of a one-to-many relationship, an intermediate table between the user and all other tables will be required, so it is predicted that 7 tables will be created, in order to normalise to the third normal form.

Comparison and Discussion of Programming Languages

3 languages were initially considered for implementing a bespoke solution in an object-oriented and event driven fashion – Java, Delphi, and C++.

Java seemed like the ideal language, for the following reasons:

- Minecraft is written in Java, therefore launching a Java application from within a Java application is fairly easy
- Memory management is done by the JVM garbage collector, so programming may be more productive
- Huge number of Java libraries

However, there were several negatives:

- Development of Java frontend is tedious, and the default framework (Swing), provides unnatural buttons. Generally, Java applications that revolve around menus and buttons provide an uncomfortable user experience
- Java applications are relatively very slow in starting, as the JVM must be initialised, and then execute the application
- Database support in Java requires the heavy use of external libraries, adding clutter to the application

This breakdown shows that Java is better suited to large, long-running tasks, with a small dependence on the user interface. However, the proposed objectives require the opposite.

Thus, Delphi and C++ were looked at. The considerations for Delphi are as follows:

- Native database support. Delphi's history shows that it has been a popular language for database-heavy programs
- Fast application start-up time as a result of compilation into machine code
- Very fast compile time
- Familiarity and competency with language
- Firemonkey frontend component suite provides very stylish components, which can create modern and clean GUIs with relatively less effort
- Extremely rapid prototyping with the Embarcadero IDE

C++ had many of the same features, but also has some fundamental differences:

- Syntactically and semantically much harder to code in
- Lots of boilerplate code needs to be written for simple tasks
- Lower competency and familiarity with language
- Faster language, but since this application's speed will largely be input/output (I/O) bound, this shouldn't matter
- No native database support
- Creating a frontend is trickier, and a non-default framework must be installed

Therefore, C++ appeared to be less advantageous than Delphi. A functional language, Haksell, was briefly looked at:

- Extremely reliable processing – a function always returns the same value
- Expressions are lazily evaluated – only evaluated when a value is required. This could improve the speed without writing optimisation code

- Still in its infancy, so there are a lack of frontend development options available
- Unfamiliarity with the functional paradigm – there is no event-driven or object oriented programming

Therefore, Delphi was chosen as the development language. Since there are many objectives of varying nature, the design must allow advanced functionality to be accessed, without intruding any users that wish to simply launch and play Minecraft. This will likely lend towards a minimalistic design, with not much deviation from the main screen. The launcher should find a way of providing maximum configurability, and clean user management, perhaps by using popup menus to apply actions to users, such as banning them or changing their time quota.

The implementation should use object-oriented programming to develop a maintainable and future-proof system. For example, classes could be created to define how a file should be downloaded, but this should be extendable and allow for perhaps downloading over a future networking protocol without changing the code that accesses the download procedure of a downloading class (polymorphism). The complexity of the implementation may increase significantly when catering for the at least the first 3 objectives, which will involve multithreaded networking code, and high-efficiency, complex string building, and recursive data structures (perhaps to index files).

This completes my analysis. I have a good understanding of my user's needs so I can now go on to the design of my proposed system.

Design

Modular Layout

The system can be broken down into the following smaller modules:

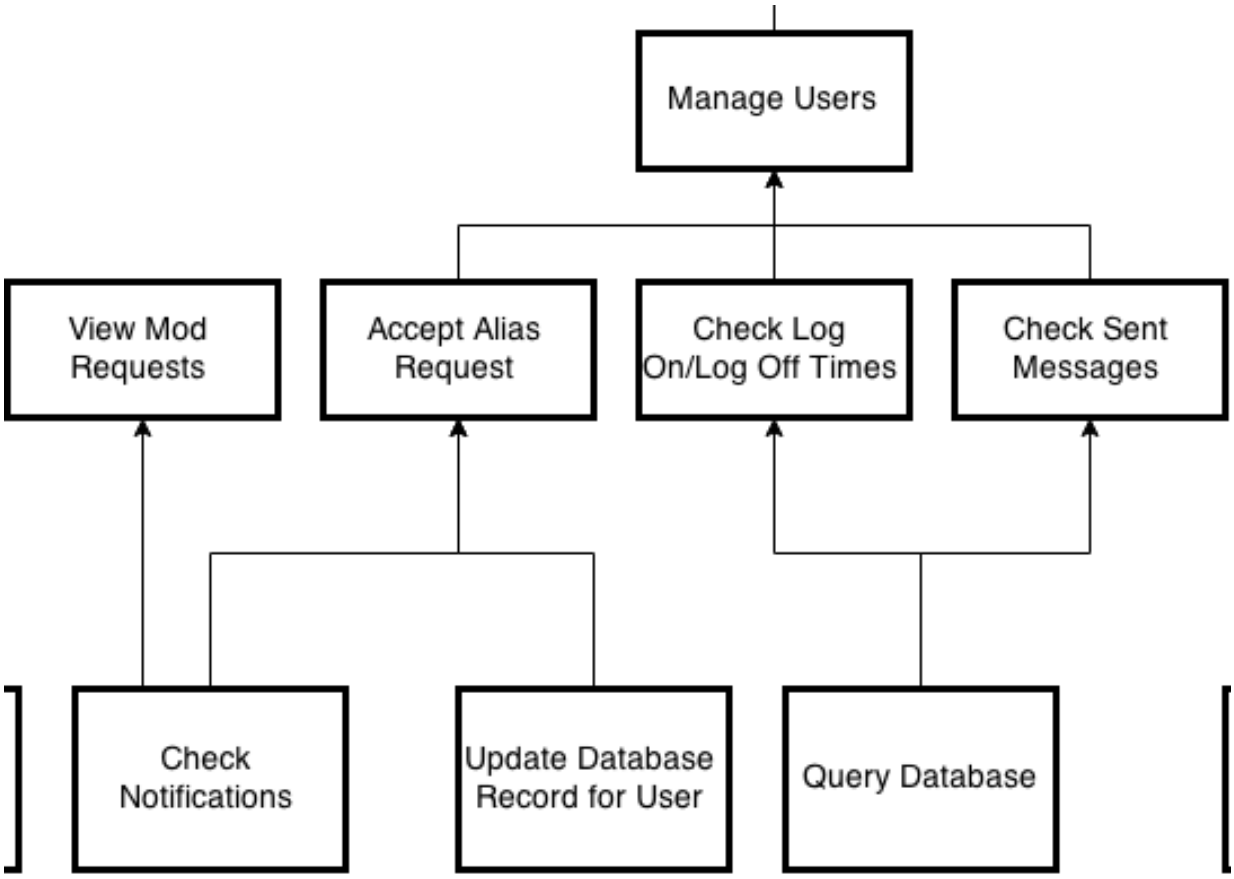
- HTTP(S) file downloader
    - This will handle any downloaders that are requested from by any other parts of the program
- List comparison
    - Compare a local list with the remote list provided by the server
- File/folder synchronisation
    - Choose which files to delete and download based on hashes
- Database storage/retrieval
    - Insert and retrieve users, sessions, sent messages, and requests
- Launch command generation
    - Generate a command based on information downloaded from the Minecraft server
- Message logging
    - Key logging another process
- Student executable generation
    - Generate a single file that embeds the settings in the executable
- Backup archive generation
    - Produce zip archives of the Minecraft directory and be able to restore it
- Minecraft Updater
    - Download a list of Minecraft versions and install any version

The hierarchy charts that follow will give a more in depth breakdown of the modular layout.

Hierarchy Chart

The hierarchy chart has been split into subsections, but all branch from “Minecraft Launcher”.

User Management



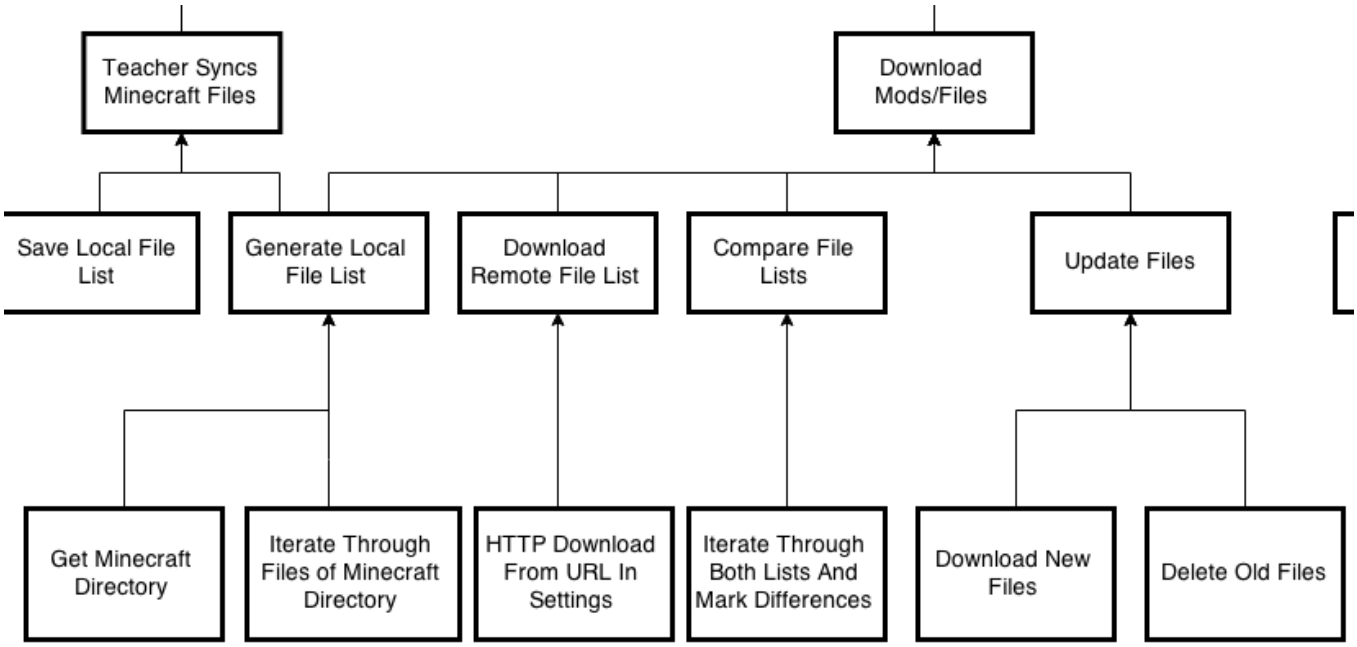
This section of the hierarchy chart breaks down the required tasks required to carry out user management.

Viewing mod requests relies on checking notifications (which simply queries the database for outstanding requests).

Accepting an alias request relies on the mechanism described above to produce a list of outstanding requests, but if accepted, it requires the functionality of updating the user’s information in the database.

Checking log on/log off times and any sent messages both rely on the ability to query the database, and return results.

Downloading Files and Syncing Files



Teacher Syncs Minecraft Files

The first task that happens is “Generate Local File List”. This is done by getting the Minecraft directory, and iterating through the files, while adding these to a list.

Save local file list simply saves the already generated file list to the hard drive, and cannot be broken down further.

Download Mods/Files

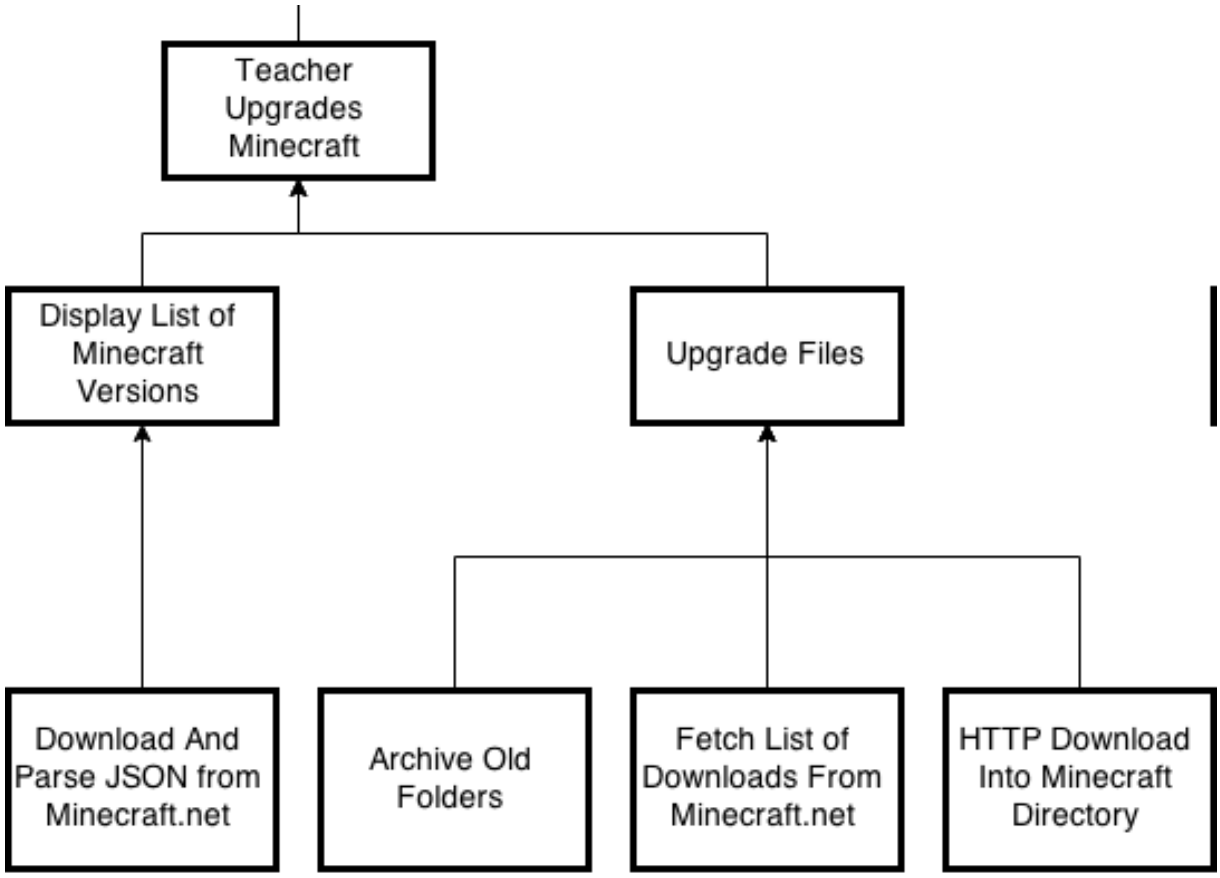
The same “Generate Local File List” task is reused (but not saved to a file as before).

Downloading the remote file list is dependent on HTTP downloading.

Compare the file lists requires iterating through both lists, and storing the differences in a new list.

Finally, updating files consists of downloading the new files (from the previously generated list) and deleting the files that are no longer required.

Upgrading Minecraft



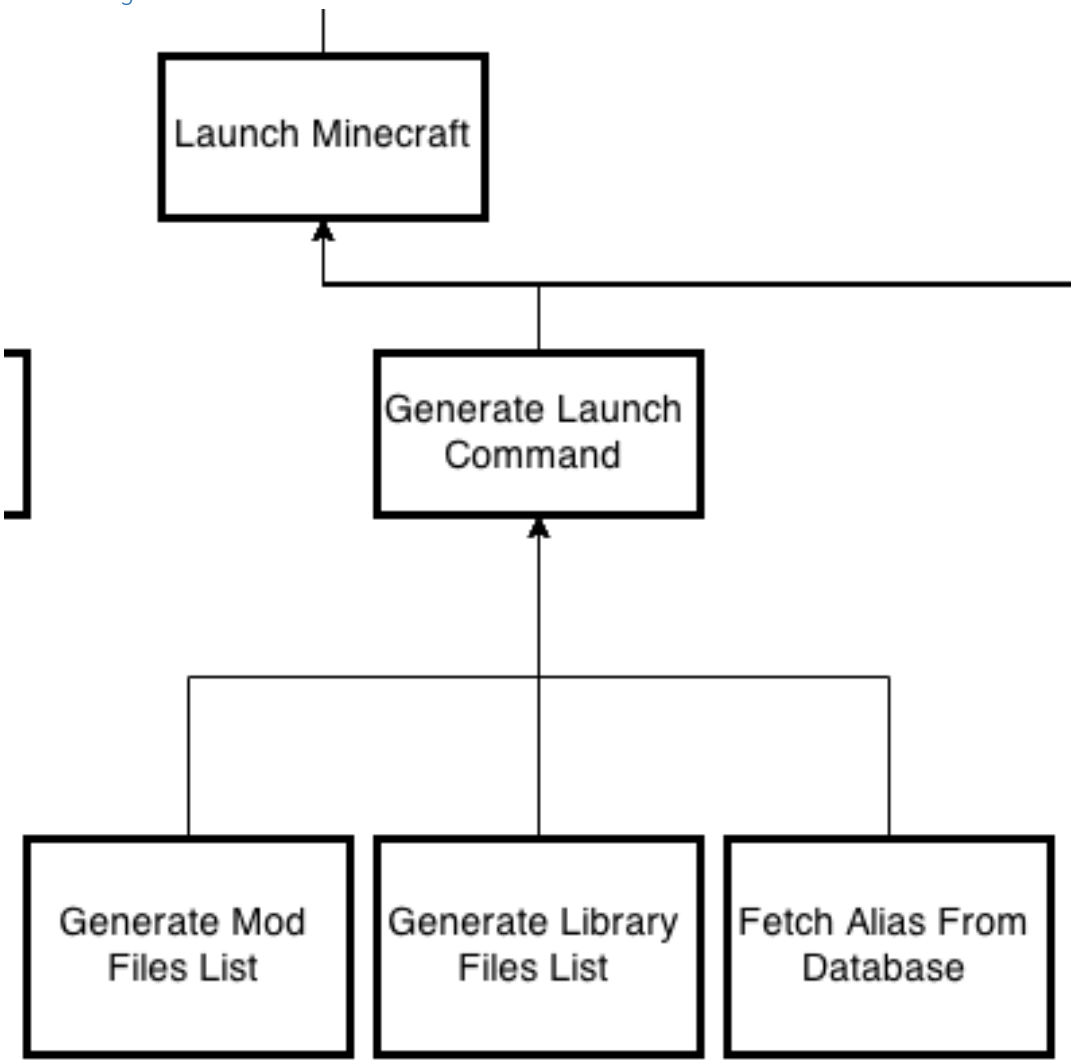
To display a list of Minecraft versions, the versions list JSON must be downloaded from Minecraft.net, and parsed (pull out the available version numbers into a list).

Upgrading the Minecraft files is composed of removing any old folders, downloading a list of files to download from Minecraft.net, and finally downloading the relevant files into the Minecraft directory.

Launching Minecraft

This will be broken up into smaller subsections, but all are part of launching Minecraft.

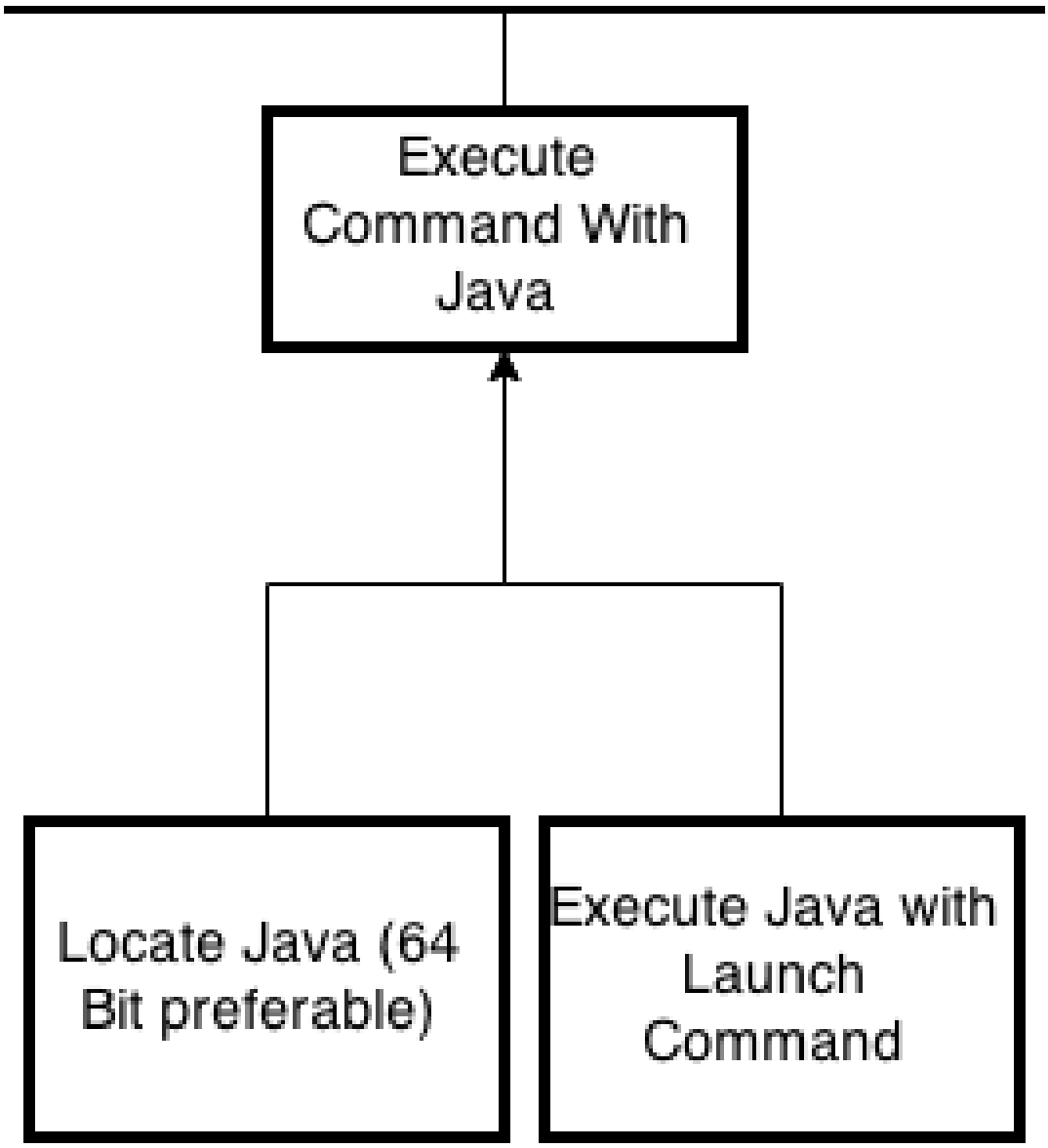
Generating the Launch Command



Generating the launch command consists of generating a list of mod files, and generating a list of library files. These are all found in the Minecraft directory, and are added to the launch command.

The alias is fetched from the database, and inserted into the launch command as the player's in-game username.

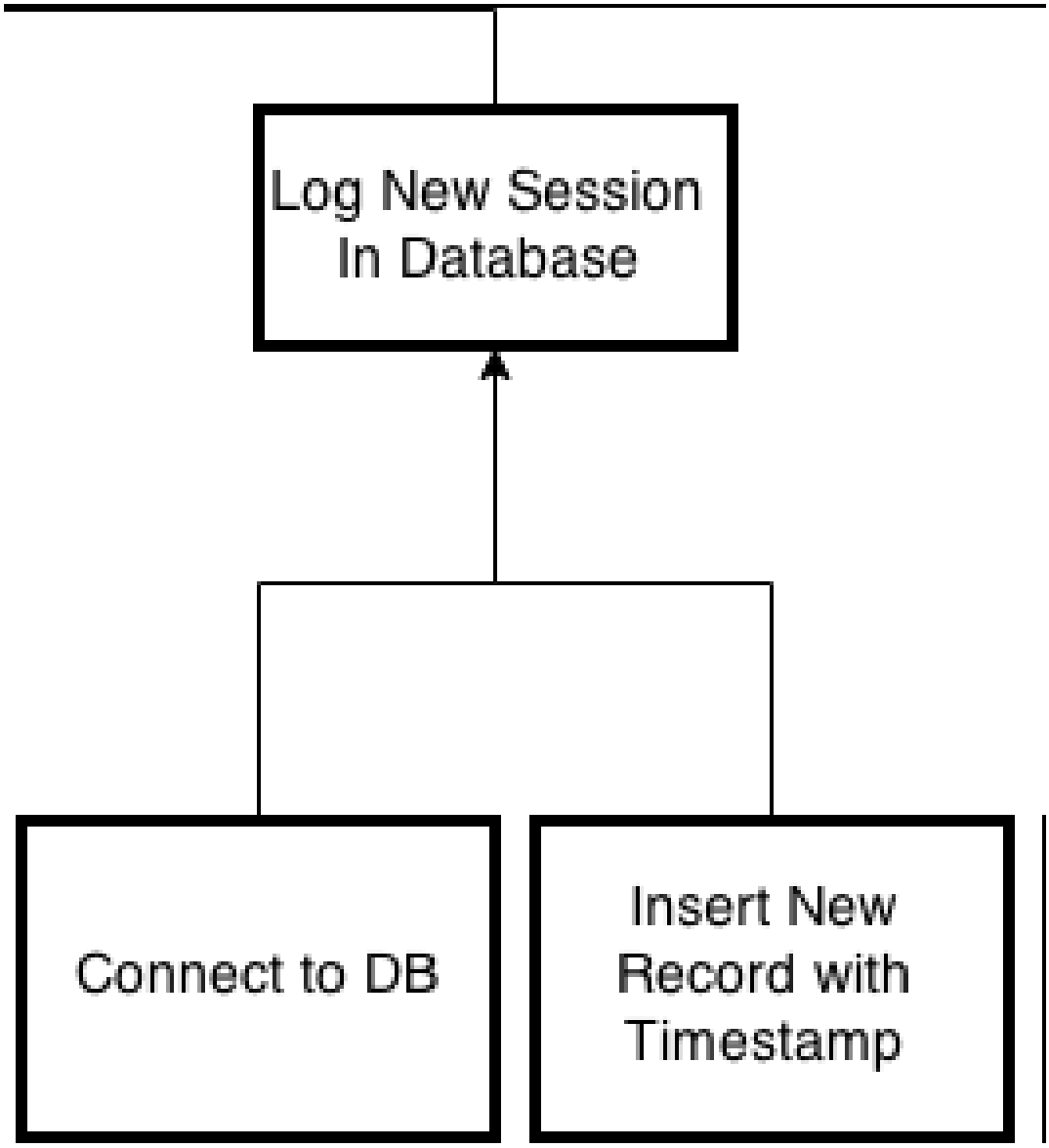
Executing Command with Java



The Java Runtime Executable (JRE) is located, using the registry. If the 64 bit version is installed, it will be used instead.

The command should then be executed with the found executable, by taking the launch command previously generated, and placing it in front of the Java executable.

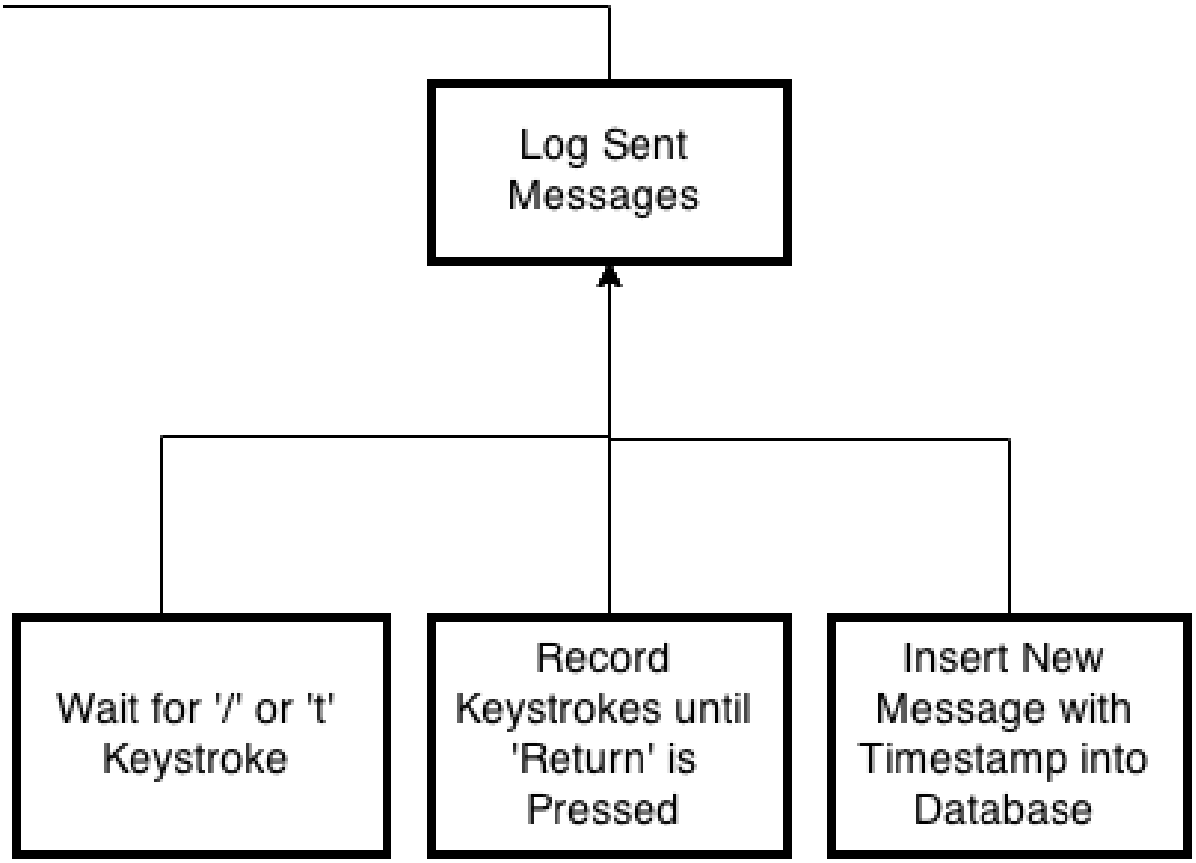
Log New Session in Database



A connection to the database must be successfully established.

A new record, containing the current time as the log on time, should be inserted into the database.

Log Sent Messages



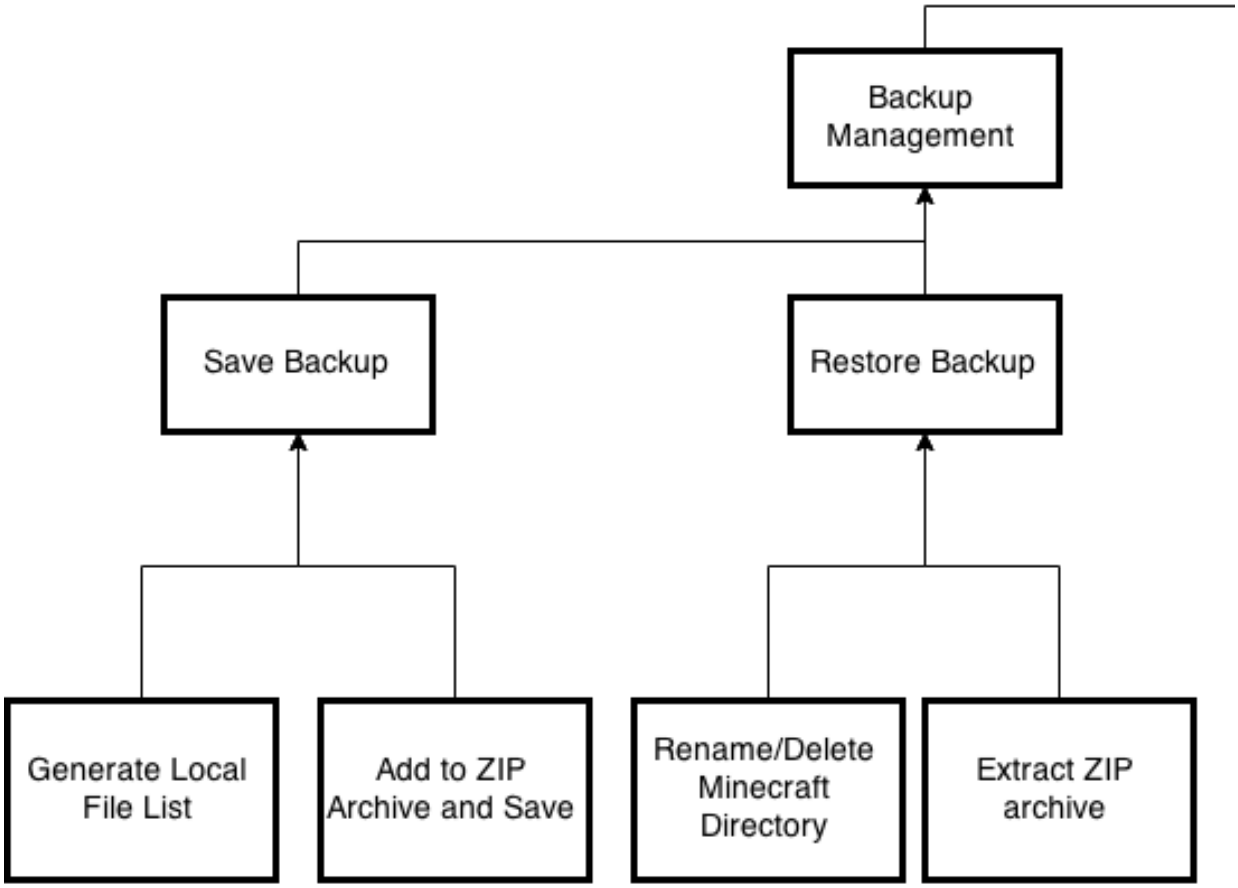
Logging sent messages is an ongoing process, and continues until the Minecraft instance has been terminated.

The application should wait until the ‘t’ or ‘/’ key has been pressed – these are the Minecraft hotkeys that allows the user to send a message or command respectively.

If the previous condition has been satisfied, any further keystrokes should be recorded until the return key (or escape key) has been pressed. The return key means that the message has been submitted (while the escape key cancels it).

The recorded message should be inserted into the database, with the current time as the time the message was sent.

Backup Management



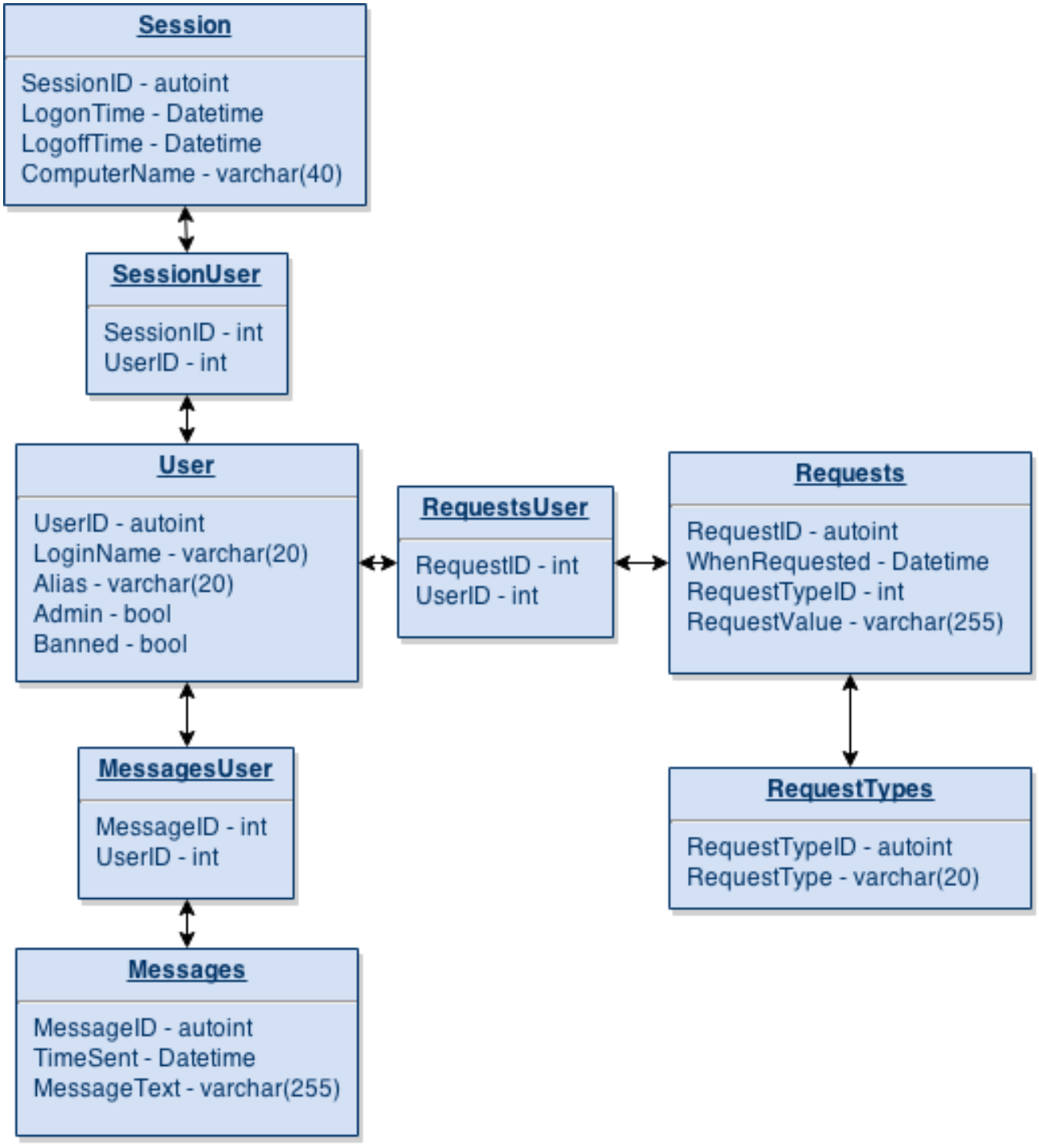
Restoring a backup requires removing the old Minecraft directory, and then extracting the contents of the backup ZIP archive into the Minecraft directory.

Saving a backup consists of generating a local file list (similarly in the syncing mods process), adding each of these files to a ZIP archive, and saving it.

Data Design

Database Design

The following database structure is proposed, using the MySQL database:



The database has been designed and normalised to 3NF. For example, all items in the "User" table only depend on the "UserID", and the only data that is repeated from this table is the primary key (UserID). Each table is in 3NF, as every non-key field is a fact about the primary key, as shown by the description of the record structure. There are also no repeating groups (1NF), or partial-key dependencies (2NF).

Description of Record Structure

MySQL Database "HMC"

Table	Name	Description	Type	Length	Size	Constraints
User	UserID	A unique reference to each user of the system	INT	N/A	4 Bytes	PRIMARY KEY, NOT NULL, AUTO_INCREMENT
	LoginName	The name of the user account the user logged on with and is identifiable by	VARCHAR	20 Characters	Maximum 21 Bytes	NOT NULL
	Alias	The in-game name of the user	VARCHAR	20 Characters	Maximum 21 Bytes	
	Quota	The time left today that the user has in-game	INT	N/A	4 Bytes	DEFAULT 30
	Admin	Defines whether the user is an administrator or not	BOOL	N/A	1 Byte	DEFAULT 0
	Banned	Indicates whether the user is banned or not	BOOL	N/A	1 Byte	DEFAULT 0
Session	SessionID	A unique reference that is created every time a user plays Minecraft	INT	N/A	4 Bytes	PRIMARY KEY, NOT NULL, AUTO_INCREMENT
	LogOnTime	The time the user began playing	DATETIME	N/A	8 Bytes	DEFAULT CURRENT_TIMESTAMP
	LogOffTime	The time the player stopped playing	DATETIME	N/A	8 Bytes	
	ComputerName	The hostname of the computer the user played on	VARCHAR	40 Characters	Maximum 41 Bytes	
SessionUser	SessionID	A link to the unique session	INT	N/A	4 Bytes	PRIMARY KEY, FOREIGN KEY, NOT NULL
	UserID	A link to the user that initiated the session	INT	N/A	4 Bytes	PRIMARY KEY, FOREIGN KEY, NOT NULL
Messages	MessageID	A unique reference to a	INT	N/A	4 Bytes	PRIMARY KEY,

		single message that a user sent in Minecraft				AUTO_INCREMENT, NOT NULL
	TimeSent	The time that the user sent the message	DATETIME	N/A	8 Bytes	DEFAULT CURRENT_TIMESTAMP
	MessageText	The contents of the message	VARCHAR	255 Characters	Maximum 257 Bytes	
MessagesUser	MessageID	A reference to the unique message	INT	N/A	4 Bytes	PRIMARY KEY, FOREIGN KEY, NOT NULL
	UserID	A reference to the user that sent the message	INT	N/A	4 Bytes	PRIMARY KEY, FOREIGN KEY, NOT NULL
RequestTypes	RequestTypeID	A unique ID of a request type	INT	N/A	4 Bytes	PRIMARY KEY, AUTO_INCREMENT, NOT NULL
	RequestType	A field indicating the type of request	VARCHAR	20 Characters	Maximum 21 Bytes	
Requests	RequestID	A unique ID representing a single request	INT	N/A	4 Bytes	PRIMARY KEY, NOT NULL, AUTO_INCREMENT
	RequestedDate	The date that the request was submitted on	DATETIME	N/A	8 Bytes	DEFAULT CURRENT_TIMESTAMP
	RequestTypeID	A link to the type of request that was submitted	INT	N/A	4 Bytes	
	RequestValue	The contents of the request	VARCHAR	255 Characters	Maximum 257 Bytes	
RequestsUser	RequestID	A link to a unique request	INT	N/A	4 Bytes	PRIMARY KEY, FOREIGN KEY, NOT NULL
	UserID	The user that submitted the request	INT	N/A	4 Bytes	PRIMARY KEY, FOREIGN KEY, NOT NULL

The database has been normalised to the third normal for (3NF), as each table has no repeating groups, no partial-key dependencies, and finally no non-key dependencies, as every field is a fact relating to only the key.

Calculations are based on the sizes below, taken from the MySQL documentation:

INT – 4 Bytes

VARCHAR – (Length + 1) Bytes if column values require 0 – 255 Bytes, (Length + 2) Bytes if values may require more than 255 Bytes

DATETIME – 8 Bytes

BOOL – 1 Byte

Assuming each VARCHAR field takes a minimum of 1 bytes to store an empty string, the minimum size of each database record is 92 Bytes.

The maximum size of each database record is 704 Bytes.

Settings Record

The settings record is either saved as a file, or embedded onto the executable when the student copy of the launcher is generated.

Field Name	Field Description	Field Type	Field Length	Field Size
CustomMinecraftDir	Stores the value of the custom Minecraft directory, if defined	String	100 Characters	200 Bytes
Xms	Stores the starting RAM of the Java Virtual Machine	Integer	N/A	4 Bytes
Xmx	Stores the maximum RAM of the Java Virtual Machine	Integer	N/A	4 Bytes
DBHost	Stores the database server hostname/address, if defined	String	100 Characters	200 Bytes
DBUsername	Stores the username required to connect to the database, if defined	String	20 Characters	40 Bytes
DBPassword	Stores the password required to connect to the database, if defined	String	20 Characters	40 Bytes
FileRepositoryURL	Stores the URL of the server Minecraft directory	String	100 Characters	200 Bytes
SecondsTillTermination	The administrator-defined time that students have to “clean up”, after their quota expires, before the Minecraft window is forcibly closed	Integer	N/A	4 Bytes
Alias	The custom alias for use if database mode is not enabled	String	20 Characters	40 Bytes
CopyTo	The file directory to automatically mirror the teacher/administrator’s copy of Minecraft to, if defined	String	100 Characters	200 Bytes
ExcludeRegex	The regular expression that is used to exclude files from being included in the server’s Minecraft directory	String	255 Characters	510 Bytes

The size calculations are based on the sizes provided by the Delphi documentation:

Integer – 4 Bytes

String – (Characters * 2) Bytes, as strings are Unicode, and thus require 2 bytes per character.

This means that the size of the record itself will be approximately 1442 Bytes. The actual file size may vary, depending on whether the record is written as a file, or embedded into the student executable.

File List Data Structure

The JavaScript Object Notation will be used to store the listing of files, and their respective Murmur2 hashes. The structure is, by definition, composite, and thus will require a recursive algorithm to extract elements and traverse the file list.

Below is an example of the contents of a potential JSON file, where if the value of each key is an object, then the key represents a folder, however if the value of the key is a string, then this is a file, where the string is the calculated Murmur2 hash for the file.

```
{
  "mods": {
    "buildcraft.jar": "5d41402abc4b2a76b9719d911017c592",
    "industrialcraft.jar": "299614d7f27cc981f3ad1f7be45c7087"
  },
  "config": {
    "buildcraft": {
      "buildcraft.txt": "3548e9edc70b6a9cc11528b74714a8ef",
      "settings.txt": "ff859d3606a46edbec08f755d48e468b"
    }
  }
}
```

Representation of the structure of the previous example:

```
▼ object {2}
   ▼ mods {2}
        buildcraft.jar : 5d41402abc4b2a76b9719d911017c592
        industrialcraft.jar : 299614d7f27cc981f3ad1f7be45c7087
   ▼ config {1}
      ▼ buildcraft {2}
           buildcraft.txt : 3548e9edc70b6a9cc11528b74714a8ef
           settings.txt : ff859d3606a46edbec08f755d48e468b
```

The JSON representation was chosen, as it is extremely lightweight, and easy for both programs and humans to read.

The file size will vary, most likely between 1-300KB, as this is highly dependent on the number of files that the teacher/administrator wishes to distribute.

Storage Requirements

Minecraft itself will take up at least 300MB. Another 50-70MB should be allocated to the launcher.

Identification of Storage Media

A copy of the launcher should be present on every computer, and there should be a predefined directory which the student accounts should have write access to on the hard drive. This path should be inputted into the launcher.

The MySQL database may run on any server, and the data will be stored on the local hard drive, and transmitted over the network when data is requested.

The file repository should be a HTTP(S) web server, and the address should be inputted into the launcher. The teacher/administrator should copy their Minecraft directory across to the server directory, after pressing the sync button (the launcher can be configured to do this automatically), which will typically be stored on hard drives. The files will be downloaded by the students' copy of the launcher, over HTTP(S), and stored on their hard drives.

Validation Required

The Murmur2 checksums of the files will be calculated once they are downloaded, and then compared. If they do not match, the program should attempt to download it only once more. The Murmur2 checksums are also used to differentiate between files that have changed, and thus re-download them. These are stored in a JSON file, as discussed earlier.

The database has length constraints, and as such, the program will be required to verify whether the data is acceptable, and will not throw an error if inserted into the database. A table of fields required validation follows, accompanied with information about the validation check itself.

Table	Name	Validation Type	On Failure
User	LoginName	Length Check – Between 1 and 20 characters	Error message prompting user
	Alias	Length Check – Between 1 and 20 characters	Error message prompting user/use LoginName
	Quota	Type Check - Integer	Error message prompting user
	Admin	Presence Check	Set to False
	Banned	Presence Check	Set to False
Session	LogOnTime	Presence Check	Set to current time
	LogOffTime	Range Check	Set to current time
Messages	MessageText	Presence Check	Ignore record insert
Requests	RequestType	Lookup Check – Check from RequestTypes Table	N/A
	RequestValue	Presence Check	Ignore record insert

Description of Measures Planned for Integrity of Data

The integrity of data is extremely important to the functioning of Minecraft – the files must be correct and complete. Hence the data structure that stores the list of files also includes the Murmur2 checksum of each file. This allows the launcher to differentiate between old and new files, and (re)download the files if the Murmur2 checksum is different to the copy of files on the server.

Description of Measures Planned for System Security

There are two components of the project that could potentially require security – the database server, and the file repository server. The launcher should allow for the optional entry of a username and password in order to connect to the database. The teacher/administrator, however, is in charge of creating the user account that can access the database (via the database server configuration). The file repository simply needs to be protected from writing to – this again is a task for the network administrators/teacher.

Planned SQL Queries

The SQL and DDL that follows was simple enough to design at this stage. These queries/commands will be embedded into the application, and executed programmatically.

DDL for Creating Tables

```
CREATE TABLE User(
        UserID INT NOT NULL AUTO_INCREMENT,
        LoginName VARCHAR(20) NOT NULL,
        Alias VARCHAR(20),
        Quota INT DEFAULT 30,
        Admin BOOL DEFAULT 0,
        Banned BOOL DEFAULT 0,
        PRIMARY KEY (UserID))
        );

CREATE TABLE Session(
        SessionID INT NOT NULL AUTO_INCREMENT,
        LogOnTime DATETIME DEFAULT CURRENT_TIMESTAMP,
        LogOffTime DATETIME,
        ComputerName VARCHAR(40),
        PRIMARY KEY (SessionID))
        );

CREATE TABLE SessionUser(
        SessionID INT NOT NULL,
        UserID INT NOT NULL,
        FOREIGN KEY (SessionID) REFERENCES Session(SessionID),
        FOREIGN KEY (UserID) REFERENCES User(UserID))
       );

CREATE TABLE Messages(
        MessageID INT NOT NULL AUTO_INCREMENT,
        TimeSent DATETIME DEFAULT CURRENT_TIMESTAMP,
        MessageText VARCHAR(255),
        PRIMARY KEY (MessageID))
      );


CREATE TABLE MessagesUser(
```

```
        MessageID INT NOT NULL,
        UserID INT NOT NULL,
        FOREIGN KEY (MessageID) REFERENCES Messages(MessageID),
        FOREIGN KEY (UserID) REFERENCES User(UserID))
        );

CREATE TABLE RequestTypes(
        RequestTypeID INT NOT NULL AUTO_INCREMENT,
        RequestType VARCHAR(20),
        PRIMARY KEY (RequestTypeID))
        );


INSERT INTO RequestTypes (RequestType)
    VALUES ("Mod"), ("Alias"), ("Ban"));

CREATE TABLE Requests(
        RequestID INT NOT NULL AUTO_INCREMENT,
        RequestedDate DATETIME DEFAULT CURRENT_TIMESTAMP,
        RequestTypeID INT,
        RequestValue VARCHAR(255),
        PRIMARY KEY (RequestID),
        FOREIGN KEY (RequestTypeID) REFERENCES RequestTypes(RequestTypeID))
        );

CREATE TABLE RequestsUser(
        RequestID INT NOT NULL,
        UserID INT NOT NULL,
        FOREIGN KEY (RequestID) REFERENCES Requests(RequestID),
        FOREIGN KEY (UserID) REFERENCES User(UserID))
        );
```

Queries

Set Administrator Mode for User ID

```
UPDATE User
SET Admin=1
WHERE UserID=:UserID
```

Set Alias for User ID

```
UPDATE User
SET Alias=:Alias
WHERE UserID=:UserID
```

Ban a User ID

```
UPDATE USER
SET Banned=1
WHERE UserID=:UserID
```

Create a New User with Login Name

```
INSERT INTO User (LoginName)
Values (:LoginName)
```

Get User ID given Login Name

```
SELECT UserID
FROM User
Where LoginName = :LoginName
```

Insert a New Message for given User ID

```
INSERT INTO Messages (MessageText)
VALUES (:MessageText);

INSERT INTO MessagesUser (MessageID, UserID)
VALUES (LAST_INSERT_ID(), :UserID);
```

Request an Alias for given User ID

```
INSERT INTO Requests (RequestTypeID, RequestValue)
VALUES (2, :Alias);

INSERT INTO RequestsUser (ReqestID, UserID)
VALUES (LAST_INSERT_ID(), :UserID);
```

Request a Mod from a given User ID

```
INSERT INTO Requests (RequestTypeID, RequestValue)
VALUES (1, :RequestValue);
INSERT INTO RequestsUser (RequestID, UserID)
VALUES (LAST_INSERT_ID(), :UserID);
```

Get all Requests and their Types

```
SELECT
User.UserID,LoginName,Requests.RequestID,RequestedDate,RequestValue,RequestType
FROM User, RequestsUser, Requests, RequestTypes
WHERE Requests.RequestID = RequestsUser.RequestID
AND RequestsUser.UserID = User.UserID
AND RequestProcessed = 0
AND RequestTypes.RequestTypeID = Requests.RequestTypeID
```

Search for Login Name, Date, Type, or Value of the Requests

```
SELECT
User.UserID,LoginName,Requests.RequestID,RequestedDate,RequestValue,RequestType
 FROM User, RequestsUser, Requests, RequestTypes
 WHERE Requests.RequestID = RequestsUser.RequestID
AND RequestsUser.UserID = User.UserID
AND RequestTypes.RequestTypeID = Requests.RequestTypeID
AND RequestProcessed = 0
AND (LoginName LIKE ":Query%"
OR RequestedDate LIKE "%:Query %"
OR RequestType LIKE "%:Query%"
OR RequestValue LIKE "%:Query%")
ORDER BY RequestedDate DESC
```

Insert a New Session for a given User ID

```
INSERT INTO Session (ComputerName)
Values(:ComputerName );

INSERT INTO SessionUser (SessionID, UserID)
VALUES (LAST_INSERT_ID(), :UserID);
```

Set Log off Time of a Session for a given Session ID

```
UPDATE Session
SET LogOffTime=NOW()
WHERE SessionID=:SessionID
```

Check if the User is an Administrator

```
SELECT Admin
FROM User
WHERE UserID=:UserID
```

Get the Alias of a given User ID

```
SELECT Alias
FROM User
WHERE UserID=:UserID
```

Check if the User is banned given a User ID

```
SELECT Banned
FROM User
WHERE UserID=:UserID
```

Get Sent Messages

```
SELECT LoginName,Alias,MessageText, TimeSent
FROM Messages, MessagesUser, User
WHERE Messages.MessageID = MessagesUser.MessageID
```

```
AND MessagesUser.UserID = User.UserID
ORDER BY TimeSent DESC
LIMIT 5000
```

Search Messages by Message Text, Time Sent, or User ID

```
SELECT LoginName, Alias, LogOnTime, LogOffTime, ComputerName
FROM Session, SessionUser, User
WHERE Session.SessionID = SessionUser.SessionID
AND SessionUser.UserID = User.UserID
AND (LoginName like "%:Query%"
OR LogOnTime like "%:Query%"
OR ComputerName like "%:Query%"
OR LogOffTime like "%:Query%")
ORDER BY LogOnTime DESC
```

Get Users, with Average Time and Session Count

```
SELECT User.UserID, LoginName, Admin, Alias, Banned, Quota,
AVG(TIME_TO_SEC(TIMEDIFF(LogOffTime, LogOnTime))/60) as Average,
COUNT(Session.SessionID) as SessionCount
FROM User, SessionUser, Session
WHERE Session.SessionID = SessionUser.SessionID
AND SessionUser.UserID = User.UserID
GROUP BY User.UserID
```

Get Sessions

```
SELECT LoginName, Alias, LogOnTime, LogOffTime, ComputerName
FROM Session, SessionUser, User
WHERE Session.SessionID = SessionUser.SessionID
AND SessionUser.UserID = User.UserID
ORDER BY LogOnTime DESC
LIMIT 5000
```

Search Sessions by Login Name, Logon Time, Logoff time, or Computer Name

```
SELECT LoginName, Alias, LogOnTime, LogOffTime, ComputerName
FROM Session, SessionUser, User
WHERE Session.SessionID = SessionUser.SessionID
AND SessionUser.UserID = User.UserID
AND (LoginName like "%:Query%"
OR LogOnTime like "%:Query%"
OR ComputerName like "%:Query%"
OR LogOffTime like "%:Query%")
ORDER BY LogOnTime DESC
```

Get Default Quota

```
SELECT DEFAULT(Quota)
FROM User
LIMIT 1
```

Set Default Quota

```
ALTER TABLE User
ALTER COLUMN Quota DROP DEFAULT;

ALTER TABLE User
ALTER COLUMN Quota SET DEFAULT :DefaultQuota;
```

Set Quota given User ID

```
UPDATE User
SET Quota=:Quota
WHERE UserID=:UserID
```

Get Quota for a given User ID

```
SELECT Quota
FROM User
WHERE UserID =:UserID
```

The Difference between the Most Recent Log off and now for a given User ID- used to reset quota

```
SELECT DateDiff(NOW(), MAX(LogOffTime))
FROM Session, SessionUser
WHERE SessionUser.UserID = :UserID
AND Session.SessionID = sessionUser.SessionID
```

Begin a Session given User ID and optional Computer Name

```
INSERT INTO Session (ComputerName)
Values(":ComputerName");

INSERT INTO SessionUser (SessionID, UserID)
VALUES (LAST_INSERT_ID(), :LocalUserID.ToString)
```

End Session given User ID

```
UPDATE Session
SET LogOffTime=NOW()
WHERE SessionID=' + SessionID.ToString
```

Get Administrator Status given User ID

```
SELECT Admin
FROM User
WHERE UserID=' + LocalUserID.ToString()
```

Identification of Processes and Suitable Algorithms for Data Transformation

JSON comparison

JavaScript objects in the JavaScript Object Notation are similar to an array, except have named indexes, known as keys. These will be accessed in pseudo-code by JSONObject['IndexName'].

The following pseudo-code will be used to implement the comparison of 2 different JSON files, and add them to a list, in a recursive fashion.

```
Procedure CompareJSON(Path)
Begin
    Extract ← False
    For keyIndex ← 0 to keys.count – 1 do
    Begin
        KeyName ← MainJSON[Keyindex].Name
        Value ← MainJSON[KeyIndex].Value
        If Exists(OtherJSON[path + keyName] then
            If not (MainJSON[path + keyName] = OtherJSON[path + keyName]) then
                Extract ← True
            else
        else Extract ← True
        if Extract = true then
            if GetDataType(Value) = JSONObject then
                CompareJSON(Path + Key  + '.')
            else
                if GetDataType(Value) = JSONArray then
                    for arrayItem in Value do CompareJSON(Path + Key + '.')
            else
                DifferenceList[DifferenceList.Length] ← Path + Key
        Extract ← False
    end
end
```

The algorithm fetches all sub-keys of the JSON object (already loaded from a file into the MainJSON variable). For every key, the name of the key, and its value is extracted. If the same key can be found in the same place in the other JSON file, but have different values, or the key cannot be found at all, the extract flag is set to true. If the extract flag is true, the subroutine checks what type of value has been extracted. If it is another JSON object, a recursive call takes place, where the same processing occurs. If it is an array, all items in the array are processed by a recursive call. This continues occurring until the base case has been met – that is the type of value is a string, so we have encountered the end of a potential tree of keys. The full path of this key should be added to a difference list, so it can be processed afterwards.

Minecraft 1.7.10 Sample JSON file

A sample of the Minecraft 1.7.10 JSON file is below. It contains information on which files to download, from where, and how to launch Minecraft.

```
{
  "id": "1.7.10",
  "time": "2014-05-14T19:29:23+02:00",
  "releaseTime": "2014-05-14T19:29:23+02:00",
  "type": "release",
  "minecraftArguments": "--username ${auth_player_name} --version ${version_name}
  "minimumLauncherVersion": 13,
  "assets": "1.7.10",
  "mainClass": "net.minecraft.client.main.Main",
  "libraries": [
    {
      "name": "com.google.code.gson:gson:2.2.4"
    },
    {
      "name": "com.mojang:authlib:1.5.16"
    },
    {
      "name": "org.apache.logging.log4j:log4j-api:2.0-beta9"
    },
    {
      "name": "org.apache.logging.log4j:log4j-core:2.0-beta9"
    },
    {
      "name": "org.lwjgl.lwjgl:lwjgl:2.9.1"
    },
    {
      "name": "org.lwjgl.lwjgl:lwjgl_util:2.9.1"
    },

      "name": "tv.twitch:twitch-external-platform:4.5",
      "rules": [
        {
          "action": "allow",
          "os": {
            "name": "windows"
          }
        }
      ],
      "natives": {
        "windows": "natives-windows-${arch}"
      },
      "extract": {
        "exclude": [
          "META-INF/"
        ]
      }
    }
  ]
}
```

This sample will be referred to in following algorithm designs. The same JSONName[KeyName] format will be used to access named key indexes in the JSON object.

Getting Library Download List from Minecraft JSON File

To play Minecraft, the correct libraries files must be downloaded, and these can be found in the relevant JSON file, as the sample shows. The pseudo-code for downloading the library files is below:

```
LibraryURL ← 'https://libraries.minecraft.net/'
For PackageJSON in JSONFile['libraries'] do
Begin
        KeyName ← PackageJSON['name']
        Package ← CopyUpToColon(KeyName)
        Package ← ReplaceCharacters(Package, '.', '/')
        KeyName ←DeleteUpToColon(KeyName)
        Name ← CopyUpToColon(KeyName)
        Name ← ReplaceCharacters(Name, '.', '/')
        KeyName ← DeleteUpToColon(Keyname)
        Version ← KeyName
DownloadURL ← LibraryURL + Package + '/' +  Name + '/' + Version + '/' + Name + '-' + Version +
'.jar
        NeedsExtracting ← False
        If Exists(PackageJSON['rules']) then
                For RuleJSON in PackageJSON['rules'] do
                Begin
                        If RuleJSON['action'] = 'disallow' then
                                If not(RulesJSON['os.name'] = 'windows')
                                        Download ← True
                        If RuleJSON['action'] = 'allow' then
                                If RulesJSON['os.name'] = 'windows' then
                                        Download ← True
                                Else
                        Else Download ← True
                End
        Else Download ← True
        If Download then DownloadFile
End
```

The algorithm loops through each item in the 'libraries' array from the JSON file, and transforms the package name into a URL of the format:

https://libraries.minecraft.net/<package>/name>/<version>/<name>-<version>.jar

Note how any '.' Characters are converted into slashes in only the name and package strings.

The algorithm checks whether there are any rules present. If not, the file is downloaded. If there are, the value 'os' key indicates that the download should or should not take place depending on the 'action' key stating 'allow' or 'disallow'. Therefore, the package "tv.twitch:twitch-external-platform:4.5" would be downloaded, from the following URL:

https://libraries.minecraft.net/ tv/twitch / twitch-external-platform /4.5/ twitch-external-platform -4.5.jar

Launching Minecraft

Launching Minecraft involves building a command string that can be executed. Libraries are added by GetLibraryListFromVersion in a similar way to the previous algorithm in terms of rules parsing and string building, so its pseudo-code has not been repeated. The same JSON file is used as in the previous example.

```
MinecraftVersion ←GetCurrentMinecraftVersion
MinecraftDir ←GetMinecraftDirectoryLocation
MinecraftJar ← MinecraftDir + '\versions\' + MinecraftVersion + '\' + MinecraftVersion + '.jar'
MainClass ← JSONFile['mainClass']
Args ← JSONFile['minecraftArguments']
ReplaceArguments(Args)
Command ← GetJavaPath + -Djava.library.path=' + MinecraftDir +'\versions\natives –cp ' +
GetLibraryList(MinecraftVersion) + ';' + MinecraftJar + ' ' + MainClass + ' ' + MinecraftArguments
ExecuteCommand(Command)

Function GetLibraryList (Version : Integer) : String
Begin
        LoadFile(Version + '.json', JSONFile)
        If Exists(JSONFile['inheritsFrom']) then
                GetLibraryList ← GetLibraryList + GetLibraryList(JSONFile['inheritsFrom'])
        GetLibraryList ← GetLibraryList + GetLibraryListFromVersion(Version)
End

Procedure ReplaceArguments(Args : String)
Begin
        Replace('${auth_player_name}, getPlayerUsername)
        Replace('${version_name}', MinecraftVersion)
End
```

The Minecraft version that is going to be launched is placed in MinecraftVersion. The Minecraft directory is held in MinecraftDir. The location of the Minecraft Java executable is in MinecraftJar, and is formed using the MinecraftDir and using location conventions to build a path. The main class is the Java main class that is to be executed by the JVM, and can be extracted from the JSON. Args hold the Minecraft arguments that are attached to the end.

ReplaceArgs simply locates any predetermined arguments, in the form of ${argument_name}, and replaces them with the corresponding value of the argument.

The command is built by concatenating:

- Java executable location
- Djava.library.path=
- List of library files. The GetLibraryList function is recursive, as some JSON files include an 'inheritsFrom' key, which indicates that libraries should also be extracted from the JSON specified by the value of 'inheritsFrom'. The value is the version number.
- ;
- Location of Minecraft .jar file
- The main class of the Minecraft .jar file that should be executed
- Any Minecraft arguments

Message Logging

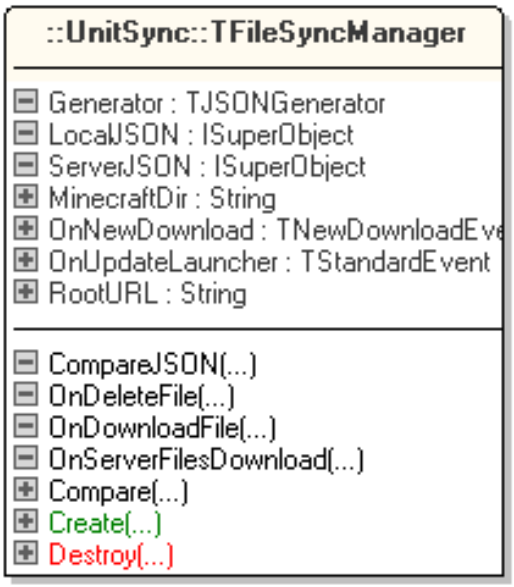
User input should be logged. The below algorithm describes how to capture messages sent in Minecraft, using a polling strategy. GetKeyState will be an operating system function that returns true or false for whether a key, passed as a parameter, is pressed or not.

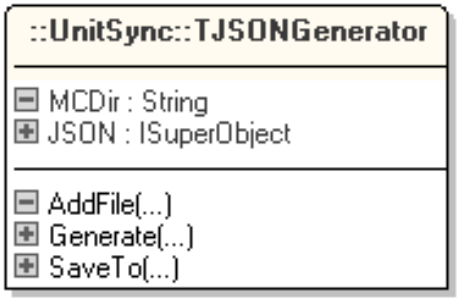
```
While Keylogging = true do
Begin
        Message ← GetMessage
        If Length(Message) > 0 then
                SubmitToDatabase(message)
End


Function GetMessage : String
Begin
        Key ← WaitForKeyPress
        If Key = 't' or Key = '/' then
                MessageInProgress ← True
        Repeat
                Key ← WaitForKeyPress
                If Key = RETURN or ESCAPE then
                        MessageInProgress ← False
                Else
                        Message ← Message + Key
                If Key = ESCAPE then
                        Message ← ''
        Until MessageInProgress = false
        GetMessage ← Message
   End


Function WaitForKeypress : Char
Begin
        Repeat
                Pause
        Until getForegroundWindowName = 'Minecraft'
         For char ← A to z do
                If getKeyState(char) = true then
                        WaitForKeyPress ← Char
End
```

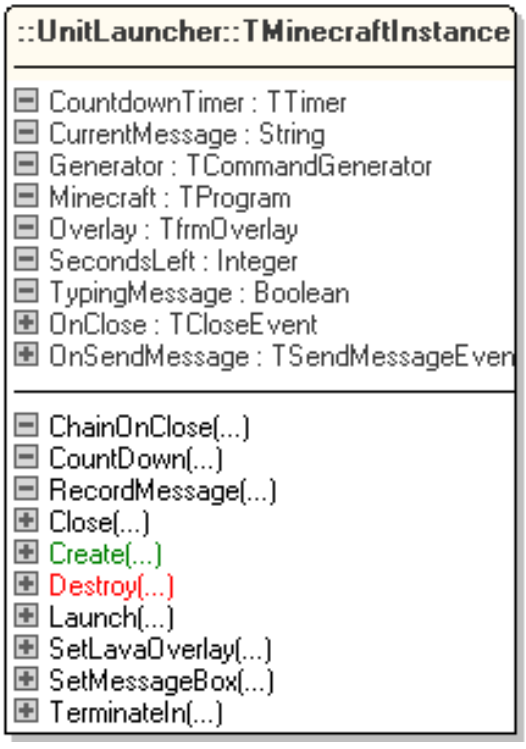
The behind the algorithm is to wait until the currently viewed window is Minecraft, and then check whether any character has been pressed by using an operating system function. This is then added to a string resembling any keys that have been typed in the Minecraft window. If the key is a 't' or '/', the Minecraft chat dialog has been summoned, and anything after this will be the message, until the return key is pressed. If the escape key has been pressed, the message is discarded.

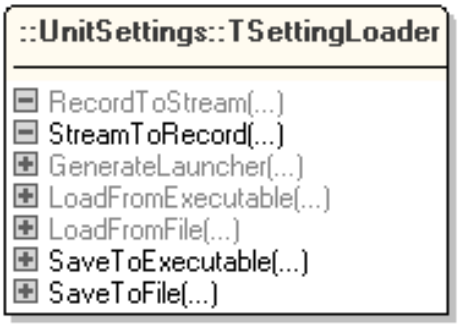
Class Diagrams

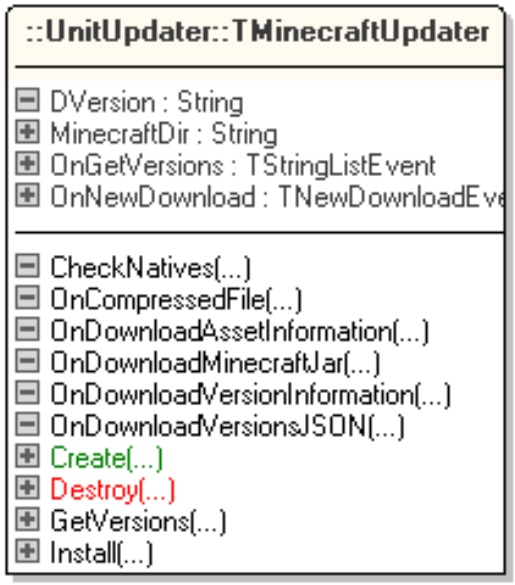
The following diagrams contain information about what each class should encapsulate, and which unit these classes should be stored in. Inheritance diagrams have been drawn separately, in the next section.

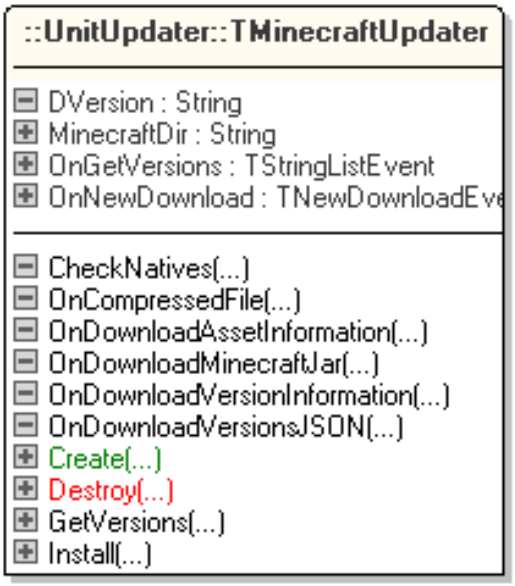


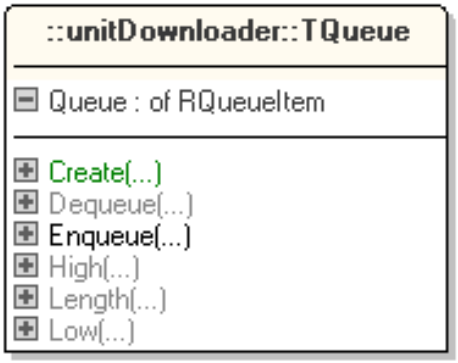


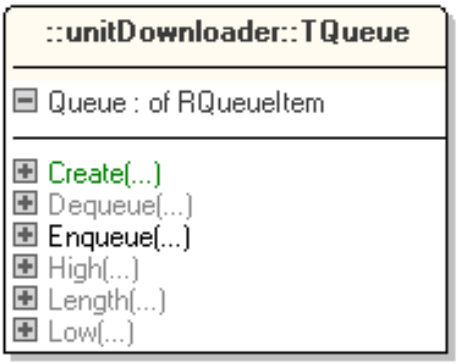


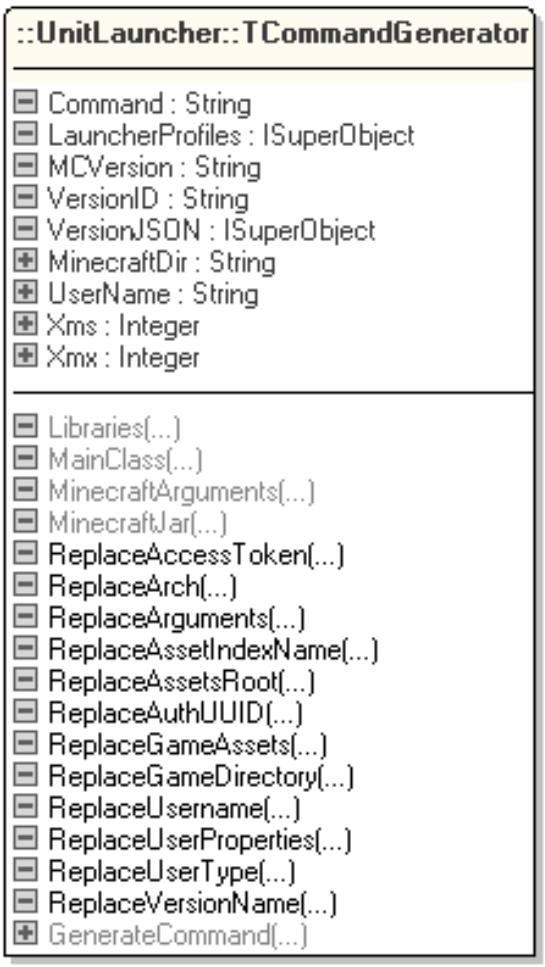


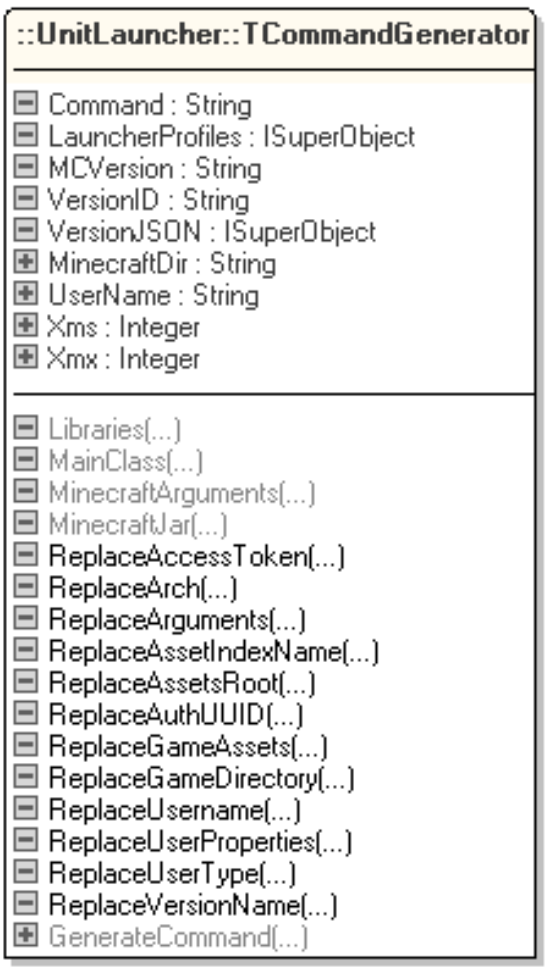

::UnitUpdater::TMinecraftUpdater
DVersion : String
MinecraftDir : String
OnGetVersions : TStringListEvent
OnNewDownload : TNewDownloadEve
CheckNatives(...)
OnCompressedFile(...)
OnDownloadAssetInformation(...)
OnDownloadMinecraftJar(...)
OnDownloadVersionInformation(...)
OnDownloadVersionsJSON(...)
Create(...)
Destroy(...)
GetVersions(...)
Install(...)

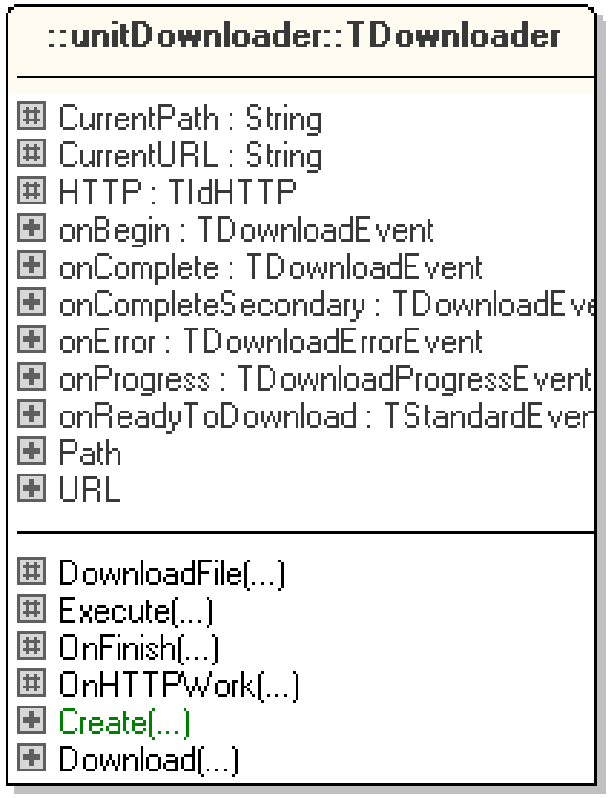

::UnitUpdater::TMinecraftUpdater
DVersion : String
MinecraftDir : String
OnGetVersions : TStringListEvent
OnNewDownload : TNewDownloadEve
CheckNatives(...)
OnCompressedFile(...)
OnDownloadAssetInformation(...)
OnDownloadMinecraftJar(...)
OnDownloadVersionInformation(...)
OnDownloadVersionsJSON(...)
Create(...)
Destroy(...)
GetVersions(...)
Install(...)

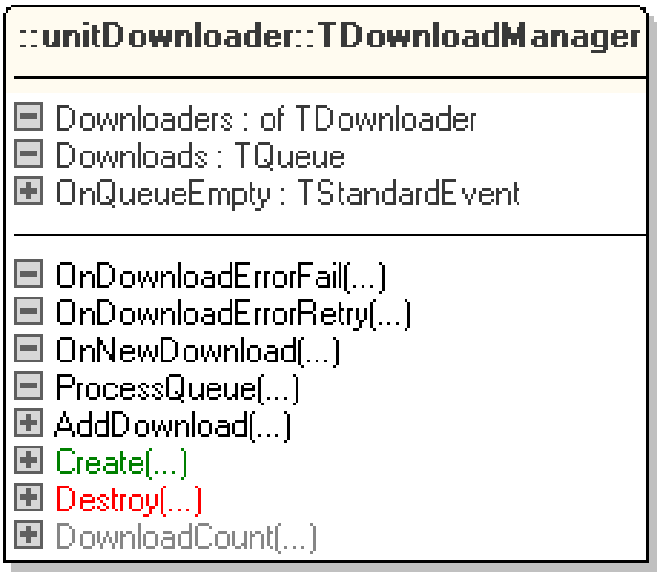

::unitDownloader::TQueue
Queue : of RQueueItem
Create(...)
Dequeue(...)
Enqueue(...)
High(...)
Length(...)
Low(...)


::unitDownloader::TQueue
Queue : of RQueueItem
Create(...)
Dequeue(...)
Enqueue(...)
High(...)
Length(...)
Low(...)

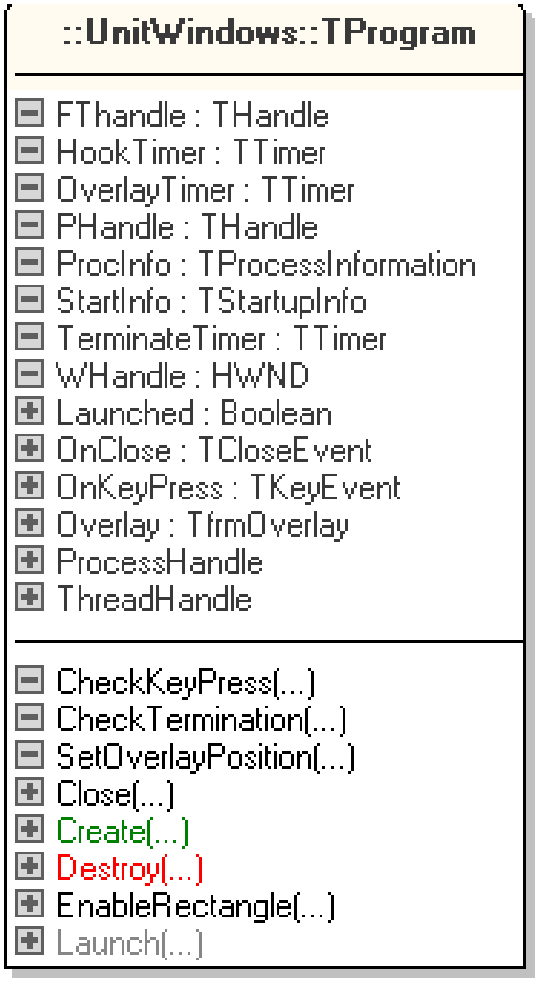

::UnitLauncher::TCommandGenerator
Command : String
LauncherProfiles : ISuperObject
MCVersion : String
VersionID : String
VersionJSON : ISuperObject
MinecraftDir : String
UserName : String
Xms : Integer
Xmx : Integer
Libraries(...)
MainClass(...)
MinecraftArguments(...)
MinecraftJar(...)
ReplaceAccessToken(...)
ReplaceArch(...)
ReplaceArguments(...)
ReplaceAssetIndexName(...)
ReplaceAssetsRoot(...)
ReplaceAuthUUID(...)
ReplaceGameAssets(...)
ReplaceGameDirectory(...)
ReplaceUsername(...)
ReplaceUserProperties(...)
ReplaceUserType(...)
ReplaceVersionName(...)
GenerateCommand(...)


::UnitLauncher::TCommandGenerator
Command : String
LauncherProfiles : ISuperObject
MCVersion : String
VersionID : String
VersionJSON : ISuperObject
MinecraftDir : String
UserName : String
Xms : Integer
Xmx : Integer
Libraries(...)
MainClass(...)
MinecraftArguments(...)
MinecraftJar(...)
ReplaceAccessToken(...)
ReplaceArch(...)
ReplaceArguments(...)
ReplaceAssetIndexName(...)
ReplaceAssetsRoot(...)
ReplaceAuthUUID(...)
ReplaceGameAssets(...)
ReplaceGameDirectory(...)
ReplaceUsername(...)
ReplaceUserProperties(...)
ReplaceUserType(...)
ReplaceVersionName(...)
GenerateCommand(...)

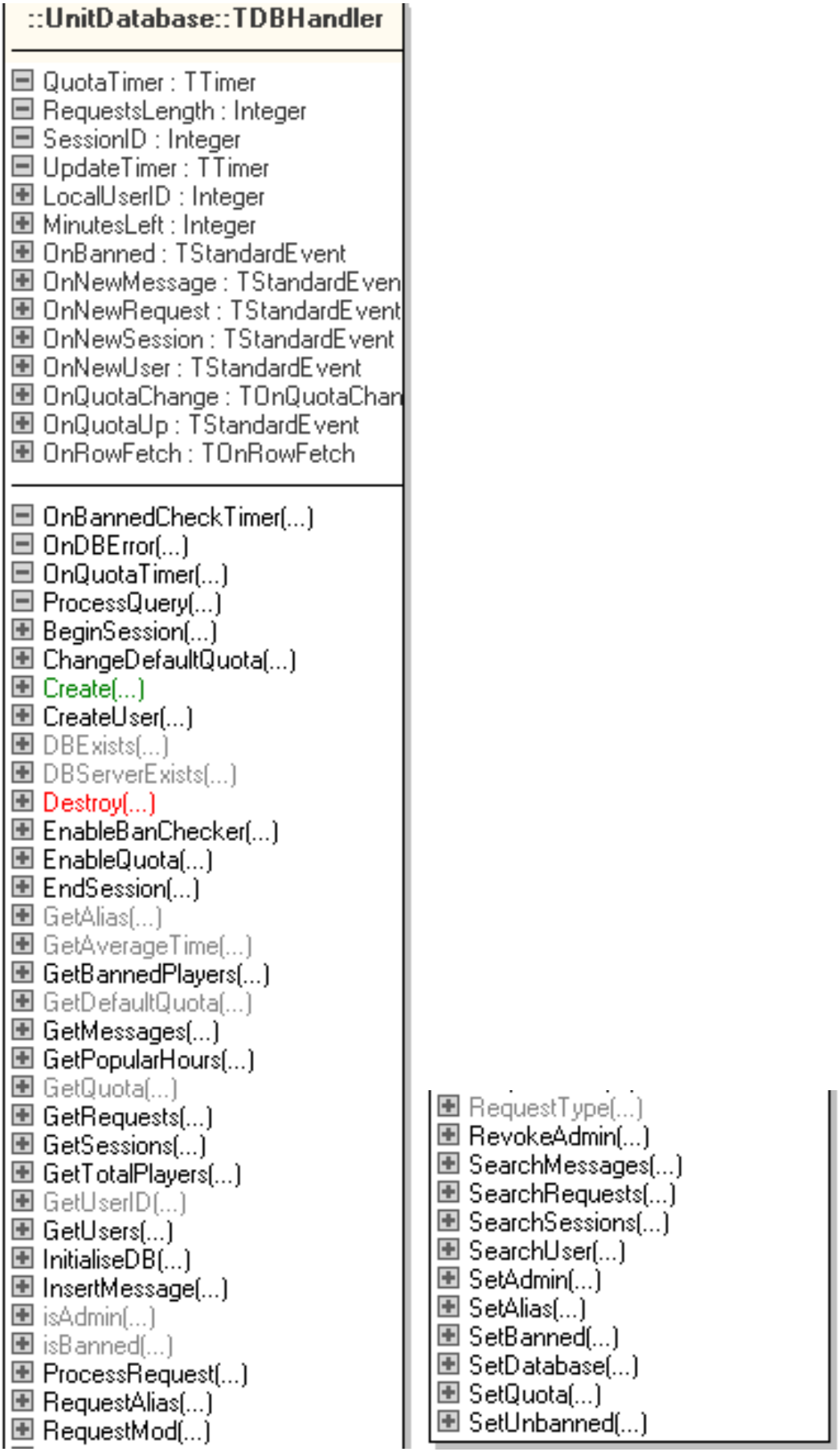

::unitDownloader::TDownloader
CurrentPath : String
CurrentURL : String
HTTP : TIdHTTP
onBegin : TDownloadEvent
onComplete : TDownloadEvent
onCompleteSecondary : TDownloadEve
onError : TDownloadErrorEvent
onProgress : TDownloadProgressEvent
onReadyToDownload : TStandardEver
Path
URL
DownloadFile(...)
Execute(...)
OnFinish(...)
OnHTTPWork(...)
Create(...)
Download(...)


::unitDownloader::TDownloadManager
Downloaders : of TDownloader
Downloads : TQueue
OnQueueEmpty : TStandardEvent
OnDownloadErrorFail(...)
OnDownloadErrorRetry(...)
OnNewDownload(...)
ProcessQueue(...)
AddDownload(...)
Create(...)
Destroy(...)
DownloadCount(...)


::UnitOverlay::TfrmOverlay
BitmapAnimation : TBitmapAnimatio
txtRemaining : TText
ChangeToLava(...)
SetText(...)

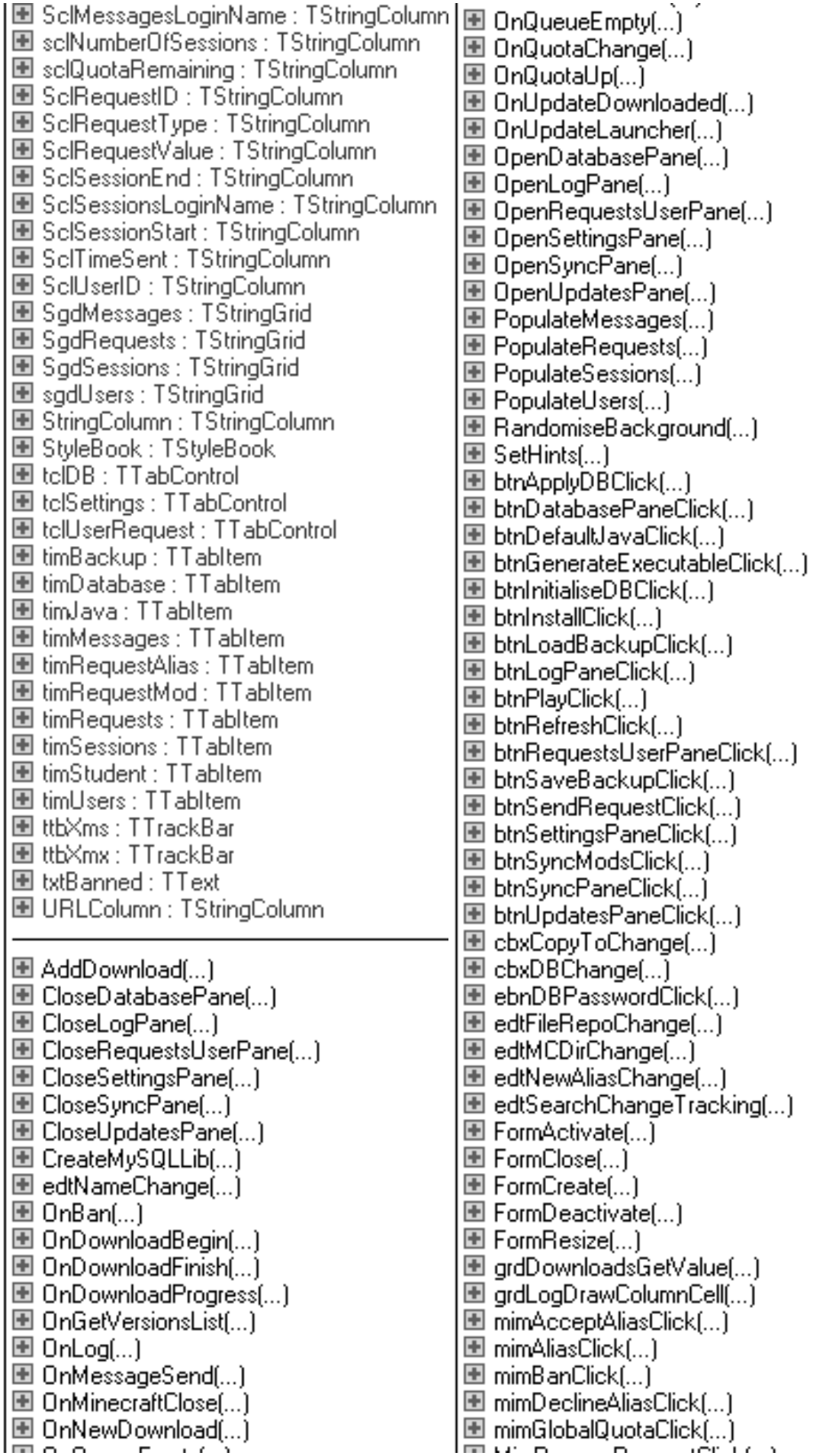

::UnitWindows::TProgram
FThandle : THandle
HookTimer : TTimer
OverlayTimer : TTimer
PHandle : THandle
ProcInfo : TProcessInformation
StartInfo : TStartupInfo
TerminateTimer : TTimer
WHandle : HWND
Launched : Boolean
OnClose : TCloseEvent
OnKeyPress : TKeyEvent
Overlay : TfrmOverlay
ProcessHandle
ThreadHandle
CheckKeyPress(...)
CheckTermination(...)
SetOverlayPosition(...)
Close(...)
Create(...)
Destroy(...)
EnableRectangle(...)
Launch(...)


::UnitLogger::TLogger
OnLog : TOnLogEvent
Error(...)
Info(...)
Warn(...)


::UnitDatabase::TDBHandler
QuotaTimer : TTimer
RequestsLength : Integer
SessionID : Integer
UpdateTimer : TTimer
LocalUserID : Integer
MinutesLeft : Integer
OnBanned : TStandardEvent
OnNewMessage : TStandardEven
OnNewRequest : TStandardEvent
OnNewSession : TStandardEvent
OnNewUser : TStandardEvent
OnQuotaChange : TOnQuotaChan
OnQuotaUp : TStandardEvent
OnRowFetch : TOnRowFetch
OnBannedCheckTimer(...)
OnDBError(...)
OnQuotaTimer(...)
ProcessQuery(...)
BeginSession(...)
ChangeDefaultQuota(...)
Create(...)
CreateUser(...)
DBExists(...)
DBServerExists(...)
Destroy(...)
EnableBanChecker(...)
EnableQuota(...)
EndSession(...)
GetAlias(...)
GetAverageTime(...)
GetBannedPlayers(...)
GetDefaultQuota(...)
GetMessages(...)
GetPopularHours(...)
GetQuota(...)
GetRequests(...)
GetSessions(...)
GetTotalPlayers(...)
GetUserID(...)
GetUsers(...)
InitialiseDB(...)
InsertMessage(...)
isAdmin(...)
isBanned(...)
ProcessRequest(...)
RequestAlias(...)
RequestMod(...)
RequestType(...)
RevokeAdmin(...)
SearchMessages(...)
SearchRequests(...)
SearchSessions(...)
SearchUser(...)
SetAdmin(...)
SetAlias(...)
SetBanned(...)
SetDatabase(...)
SetQuota(...)
SetUnbanned(...)

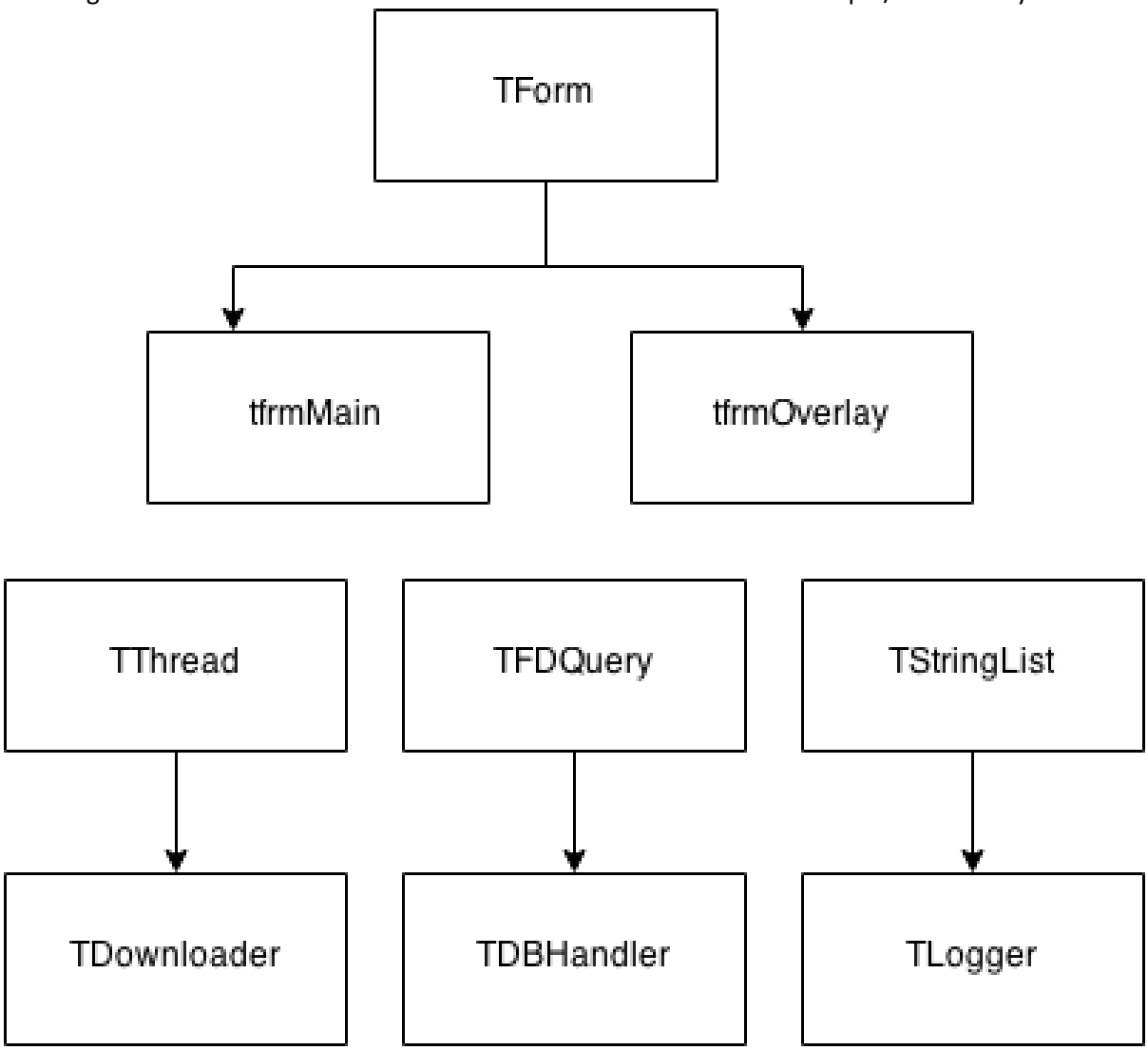

::UnitMain::TfrmMain
Downloads : of String
DProgress : of Integer
AniBusy : TAniIndicator
BindingsList : TBindingsList
BlurEffectBanned : TBlurEffect
btnApplyDB : TButton
btnDatabasePane : TButton
btnDefaultJava : TButton
btnGenerateExecutable : TButton
btnInitialiseDB : TButton
btnInstall : TButton
btnLoadBackup : TButton
btnLogPane : TButton
btnPlay : TButton
btnRefresh : TButton
btnRequestsUserPane : TButton
btnSaveBackup : TButton
btnSendRequest : TButton
btnSettingsPane : TButton
btnSyncMods : TButton
btnSyncPane : TButton
btnUpdatesPane : TButton
cbxCopyTo : TCheckBox
cbxDB : TCheckBox
cmbVersions : TComboBox
ebnDBPassword : TEditButton
edtDBPassword : TEdit
edtDBURL : TEdit
edtDBUsername : TEdit
edtFileRepo : TEdit
edtMCDir : TEdit
edtModRequest : TEdit
edtName : TEdit
edtNewAlias : TEdit
edtSearch : TEdit
FDGUIxAsyncExecuteDialog : TFDGUIxAsyncExecuteDial
FDGUIxWaitCursor : TFDGUIxWaitCursor
FDPhysMySQLDriverLink : TFDPhysMySQLDriverLink
FloatAnimationBanned : TFloatAnimation
grdDownloads : TGrid
grdLog : TStringGrid
imgBackground : TImage
lblBecause : TLabel
lblCustomMCDirInfo : TLabel
lblDBPassword : TLabel
lblDBURL : TLabel
lblDBUsername : TLabel
lblIWant : TLabel
lblModRepoInfo : TLabel
lblQuota : TLabel
lblQuotaTTT : TLabel


lblReborn : TLabel
lblRegexExclude : TLabel
lblVersion : TLabel
lblXms : TLabel
lblXmsInfo : TLabel
lblXmsMB : TLabel
lblXmx : TLabel
lblXmxInfo : TLabel
lblXmxMB : TLabel
LinkControlToPropertyEnabled : TLinkControlToProperty
LinkControlToPropertyEnabled2 : TLinkControlToProperty
LinkControlToPropertyEnabled3 : TLinkControlToProperty
LinkControlToPropertyEnabled4 : TLinkControlToProperty
LinkControlToPropertyEnabled5 : TLinkControlToProperty
LinkControlToPropertyEnabled6 : TLinkControlToProperty
LinkControlToPropertyEnabled7 : TLinkControlToProperty
LinkControlToPropertyEnabled8 : TLinkControlToProperty
mimAcceptAlias : TMenuItem
mimAlias : TMenuItem
mimBan : TMenuItem
mimDeclineAlias : TMenuItem
mimGlobalQuota : TMenuItem
MimProcessRequest : TMenuItem
mimQuota : TMenuItem
mimRevokeAdmin : TMenuItem
mimSetAdmin : TMenuItem
mimUnban : TMenuItem
mmoBecause : TMemo
mmoRegex : TMemo
OpenBackupDialog : TOpenDialog
OpenDialogSync : TOpenDialog
pnlDB : TPanel
pnlLog : TPanel
pnlRequestUser : TPanel
pnlSettings : TPanel
pnlSync : TPanel
pnlUpdates : TPanel
PopupMenuDB : TPopupMenu
PopupMenuRequests : TPopupMenu
ProgressColumn : TProgressColumn
SaveBackupDialog : TSaveDialog
SaveExeDialog : TSaveDialog
sbxQuotaTTT : TSpinBox
sclAdmin : TStringColumn
sclAlias : TStringColumn
sclAverageTime : TStringColumn
sclBanned : TStringColumn
sclComputerName : TStringColumn
SclDateRequest : TStringColumn
sclHiddenID : TStringColumn
sclLoginName : TStringColumn
SclLoginNameRequests : TStringColumn
SclMessage : TStringColumn


SclMessagesLoginName : TStringColumn
sclNumberOfSessions : TStringColumn
sclQuotaRemaining : TStringColumn
SclRequestID : TStringColumn
SclRequestType : TStringColumn
SclRequestValue : TStringColumn
SclSessionEnd : TStringColumn
SclSessionsLoginName : TStringColumn
SclSessionStart : TStringColumn
SclTimeSent : TStringColumn
SclUserID : TStringColumn
SgdMessages : TStringGrid
SgdRequests : TStringGrid
SgdSessions : TStringGrid
sgdUsers : TStringGrid
StringColumn : TStringColumn
StyleBook : TStyleBook
tclDB : TTabControl
tclSettings : TTabControl
tclUserRequest : TTabControl
timBackup : TTabItem
timDatabase : TTabItem
timJava : TTabItem
timMessages : TTabItem
timRequestAlias : TTabItem
timRequestMod : TTabItem
timRequests : TTabItem
timSessions : TTabItem
timStudent : TTabItem
timUsers : TTabItem
ttbXms : TTrackBar
ttbXmx : TTrackBar
txtBanned : TText
URLColumn : TStringColumn
AddDownload(...)
CloseDatabasePane(...)
CloseLogPane(...)
CloseRequestsUserPane(...)
CloseSettingsPane(...)
CloseSyncPane(...)
CloseUpdatesPane(...)
CreateMySQLLib(...)
edtNameChange(...)
OnBan(...)
OnDownloadBegin(...)
OnDownloadFinish(...)
OnDownloadProgress(...)
OnGetVersionsList(...)
OnLog(...)
OnMessageSend(...)
OnMinecraftClose(...)
OnNewDownload(...)
OnQueueEmpty(...)
OnQuotaChange(...)
OnQuotaUp(...)
OnUpdateDownloaded(...)
OnUpdateLauncher(...)
OpenDatabasePane(...)
OpenLogPane(...)
OpenRequestsUserPane(...)
OpenSettingsPane(...)
OpenSyncPane(...)
OpenUpdatesPane(...)
PopulateMessages(...)
PopulateRequests(...)
PopulateSessions(...)
PopulateUsers(...)
RandomiseBackground(...)
SetHints(...)
btnApplyDBClick(...)
btnDatabasePaneClick(...)
btnDefaultJavaClick(...)
btnGenerateExecutableClick(...)
btnInitialiseDBClick(...)
btnInstallClick(...)
btnLoadBackupClick(...)
btnLogPaneClick(...)
btnPlayClick(...)
btnRefreshClick(...)
btnRequestsUserPaneClick(...)
btnSaveBackupClick(...)
btnSendRequestClick(...)
btnSettingsPaneClick(...)
btnSyncModsClick(...)
btnSyncPaneClick(...)
btnUpdatesPaneClick(...)
cbxCopyToChange(...)
cbxDBChange(...)
ebnDBPasswordClick(...)
edtFileRepoChange(...)
edtMCDirChange(...)
edtNewAliasChange(...)
edtSearchChangeTracking(...)
FormActivate(...)
FormClose(...)
FormCreate(...)
FormDeactivate(...)
FormResize(...)
grdDownloadsGetValue(...)
grdLogDrawColumnCell(...)
mimAcceptAliasClick(...)
mimAliasClick(...)
mimBanClick(...)
mimDeclineAliasClick(...)
mimGlobalQuotaClick(...)

Inheritance Diagrams

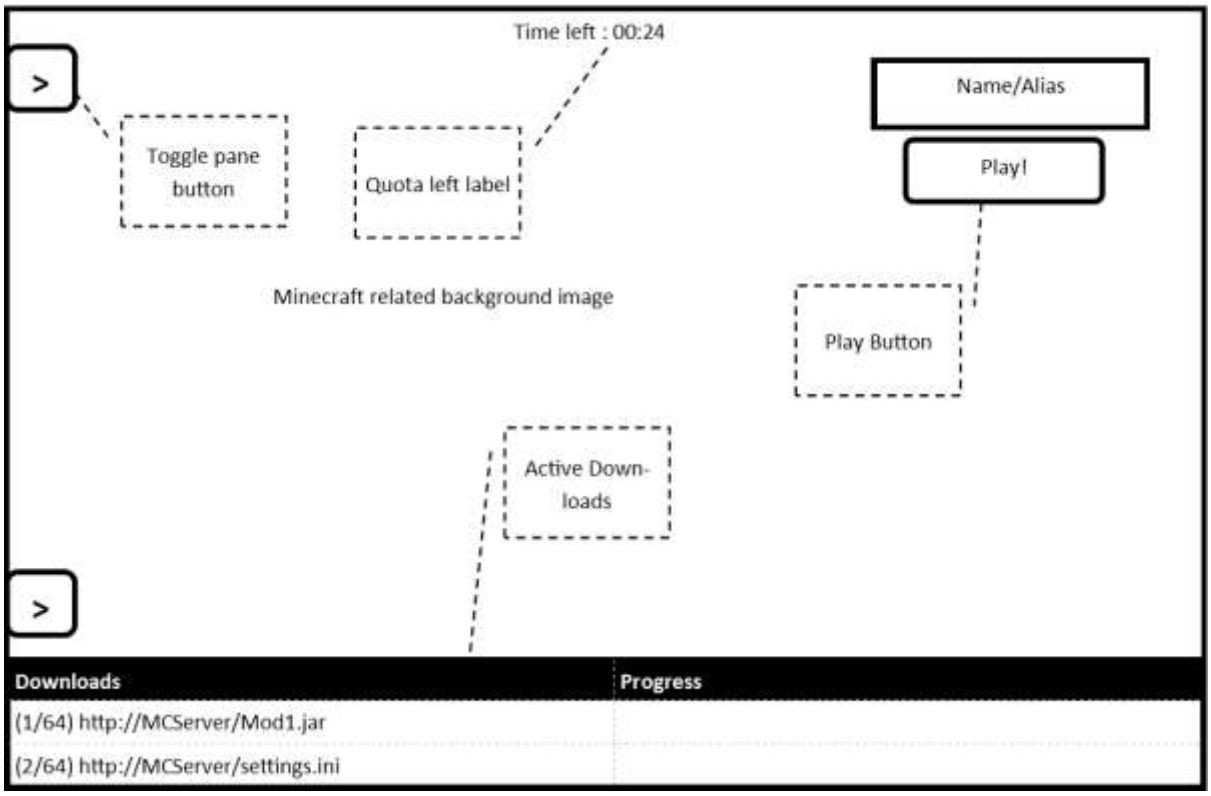
The diagrams below show the inheritance of user-defined classes from Delphi/Firemonkey classes:



Prototyping – User Interface Design

Main Screen

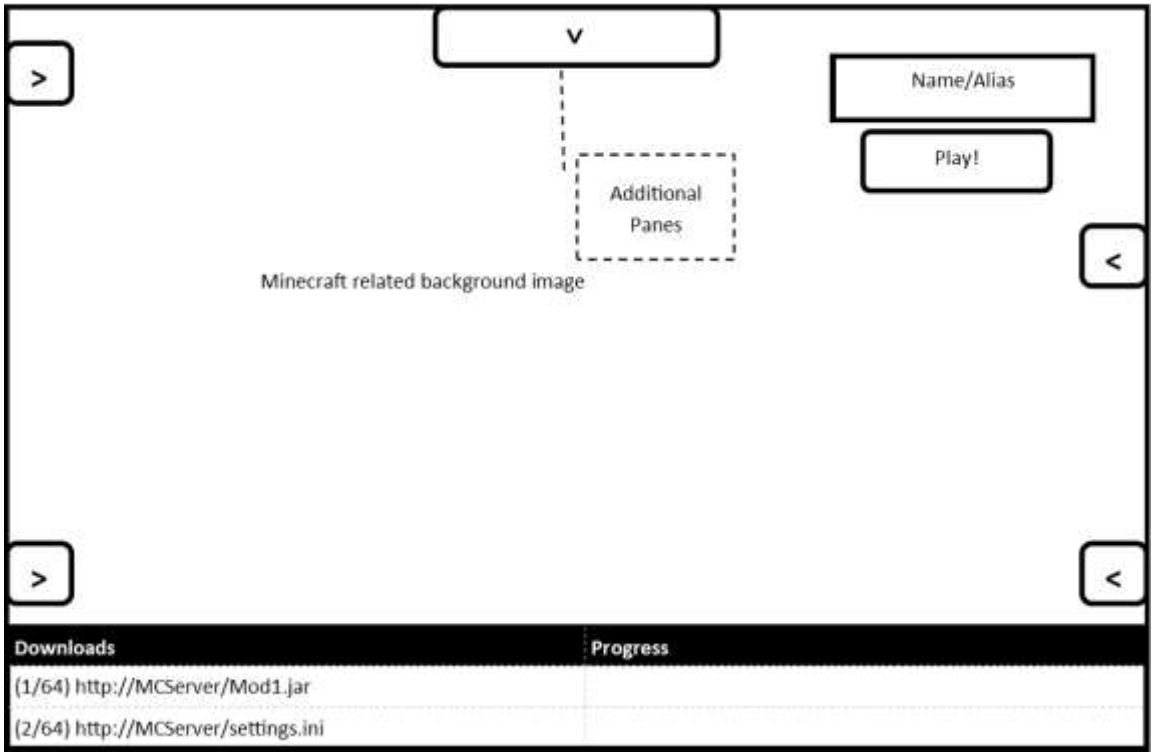
Main Screen (Student Point of View)



The functionality is fairly self-explanatory – press the play button to play Minecraft, and press any of the arrow toggle buttons to open or close a pane.

Main Screen (Student Point of View) if Banned

Main Screen (Teacher Point of View)

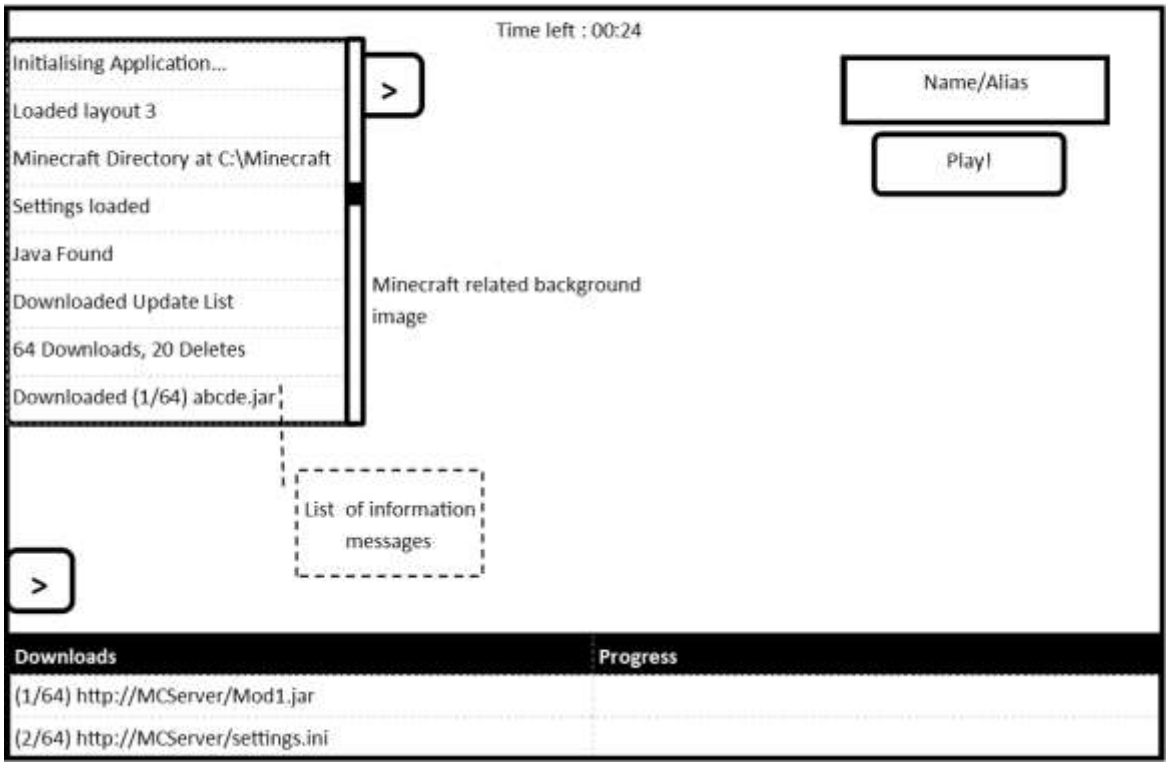


The teacher will see a different set of toggle arrows, as they should have different panes available.

Log Pane

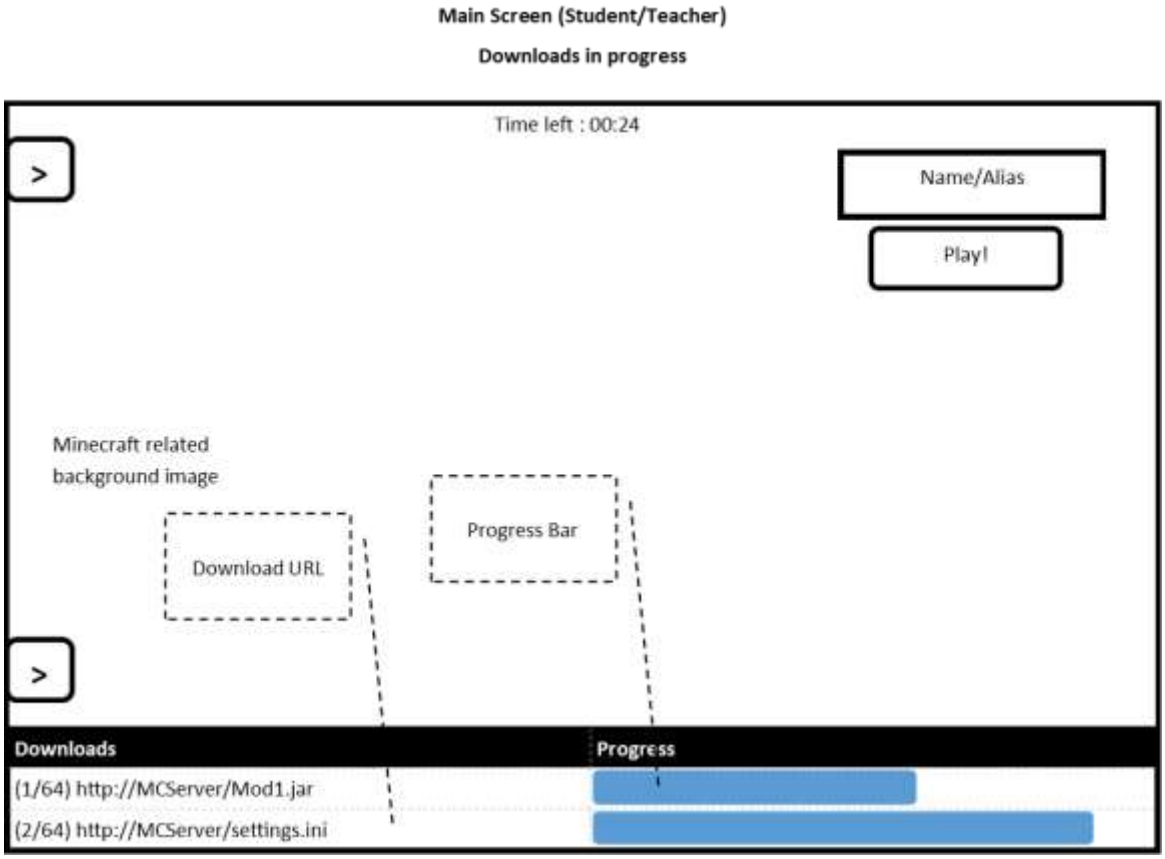
Main Screen (Student /Teacher)

Log Pane



The log pane contains a list of messages from the application. This may help debug any errors.

Downloads Pane

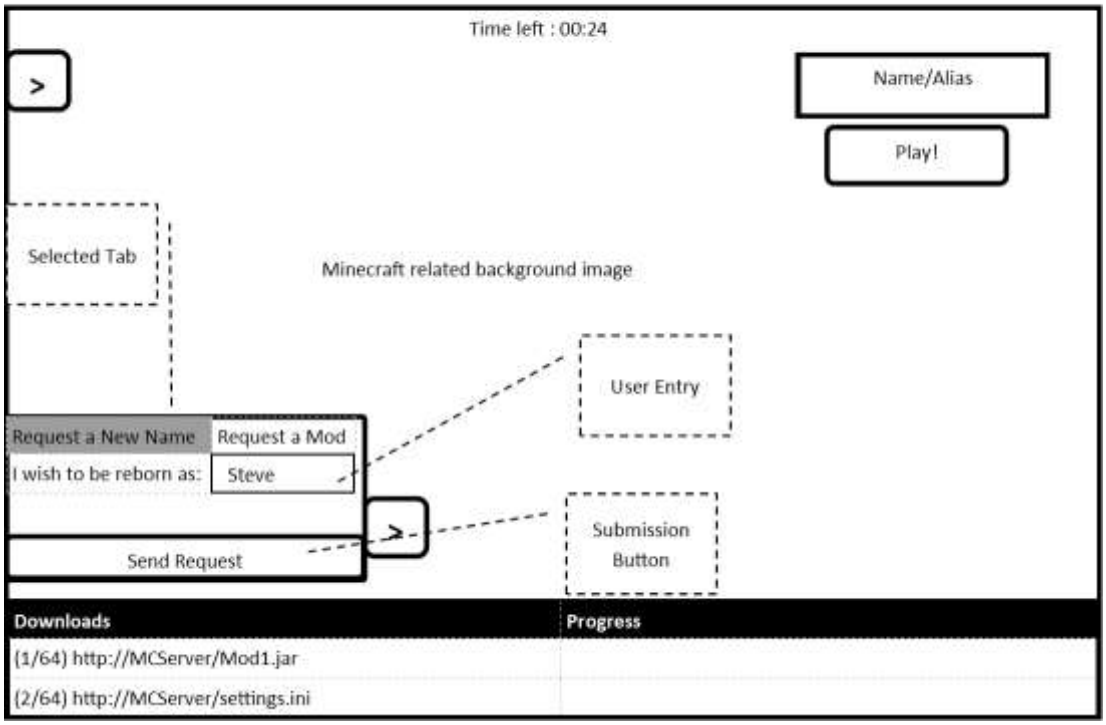


The downloads pane shows any ongoing downloads, their URL and their progress. These downloads happen simultaneously.

Requests Pane

Main Screen (Student POV)

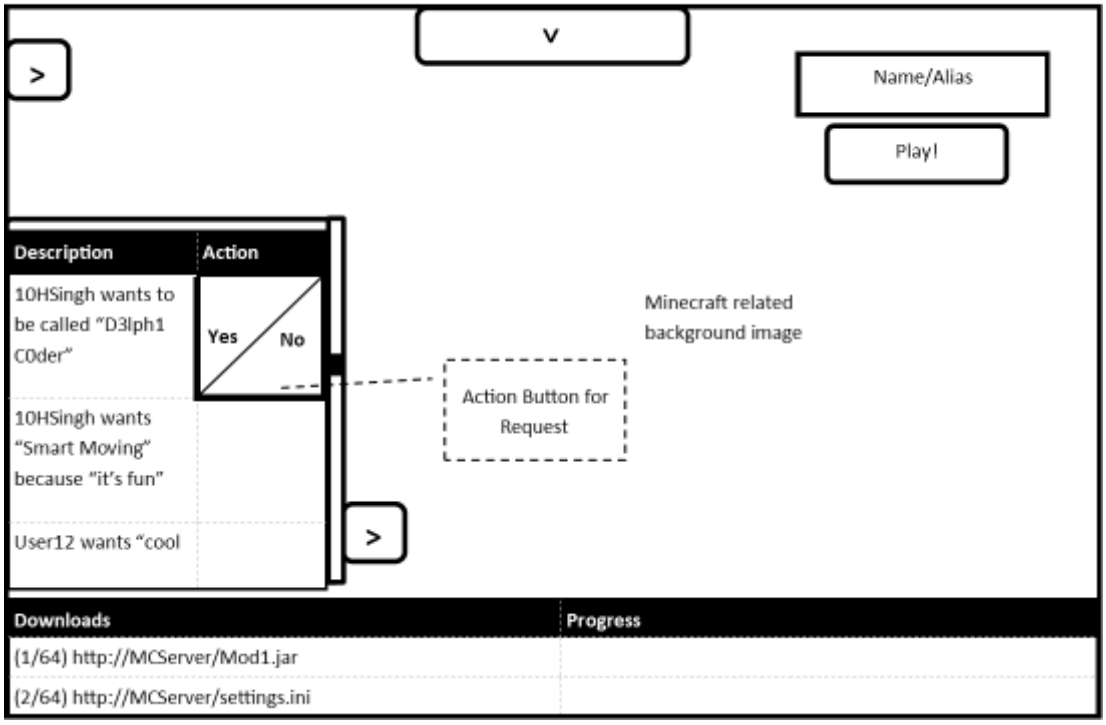
Request Pane—Request Alias



The student types in the desired name, and presses the send request button. An informative dialog suggesting that this has happened/failed should appear.

Main Screen (Teacher)

Alias and Mod Request Management Pane

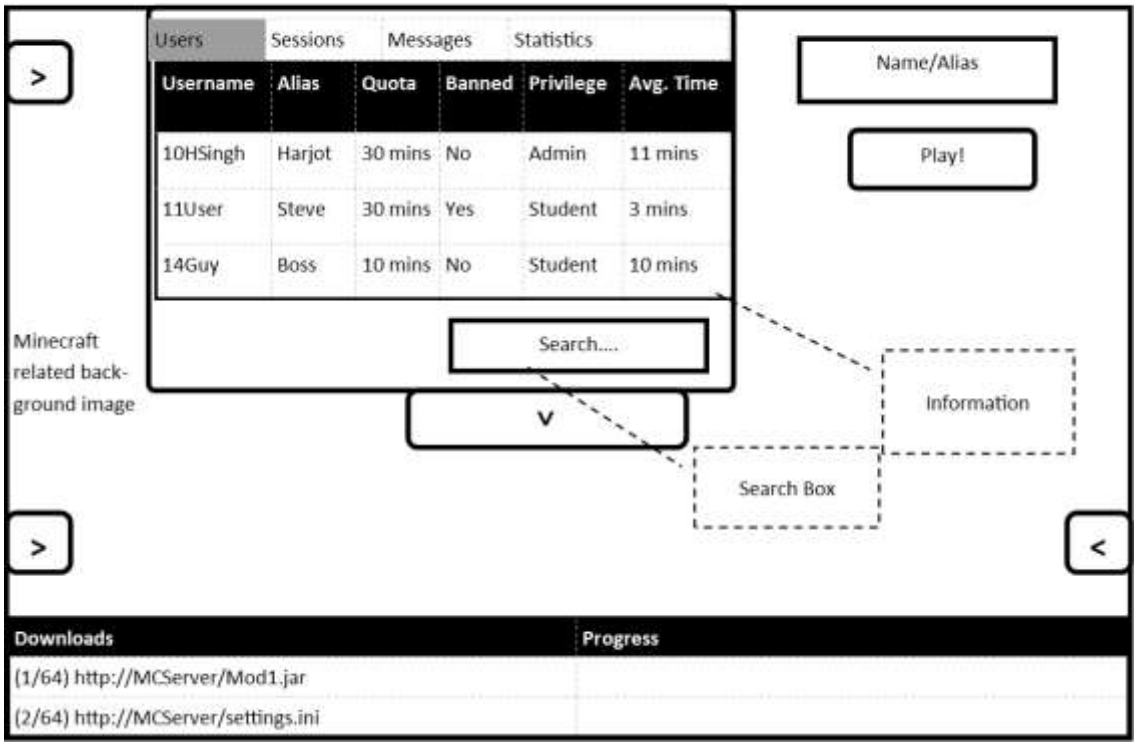


A list of the requests should appear in the teachers request pane, and should be actionable.

Database Pane

Main Screen (Teacher POV)

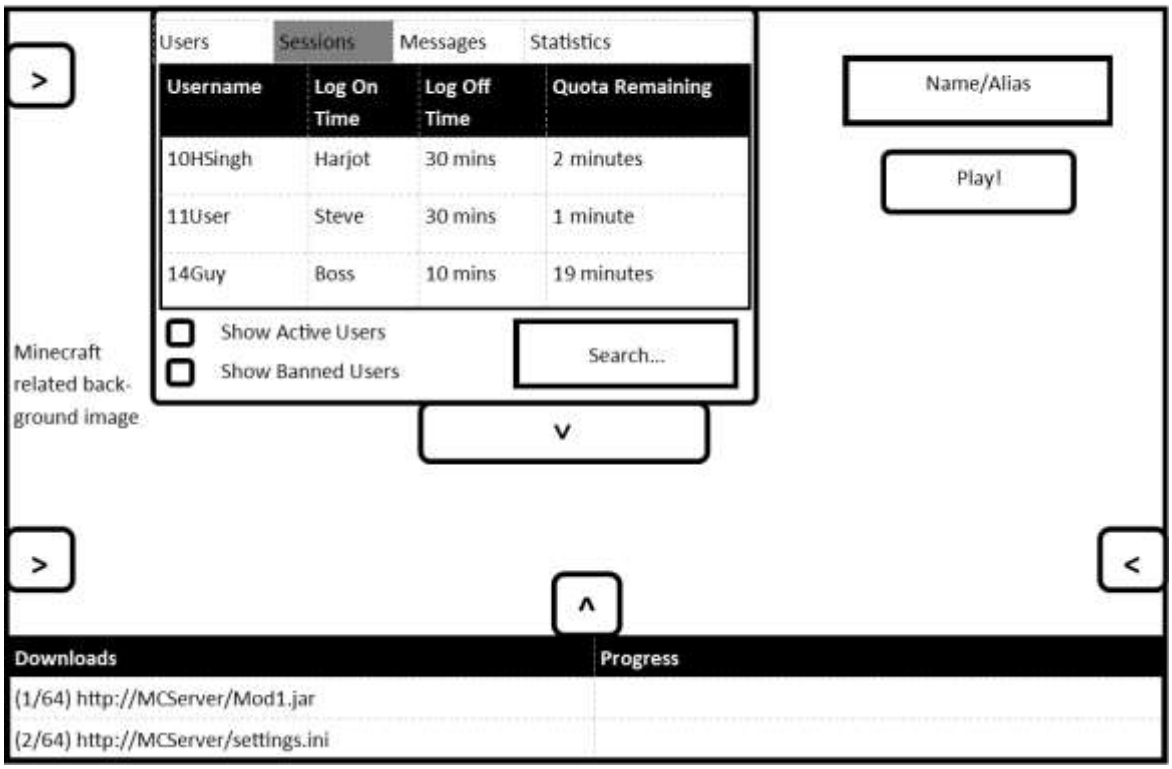
Database Pane—User Management



The teacher can view any database-related functionality, and carry out actions by right clicking on a username, where a popup menu should appear.

Main Screen (Teacher POV)

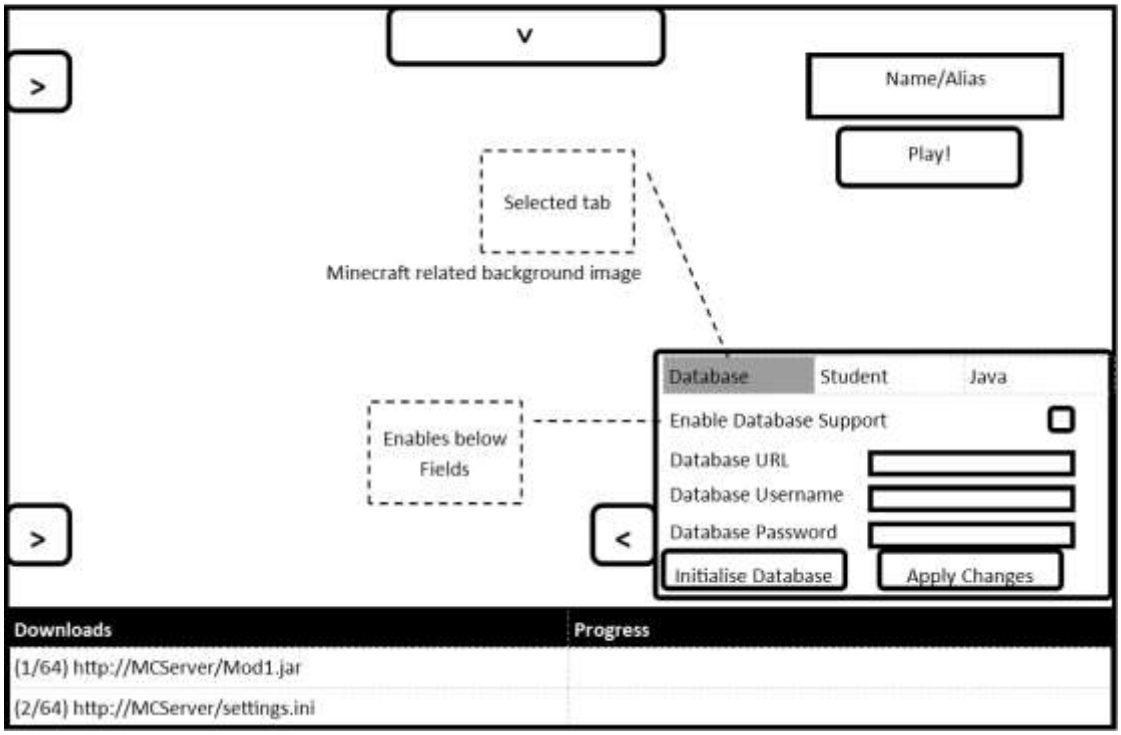
Database Pane—Session Management



Settings Pane

Main Screen (Teacher POV)

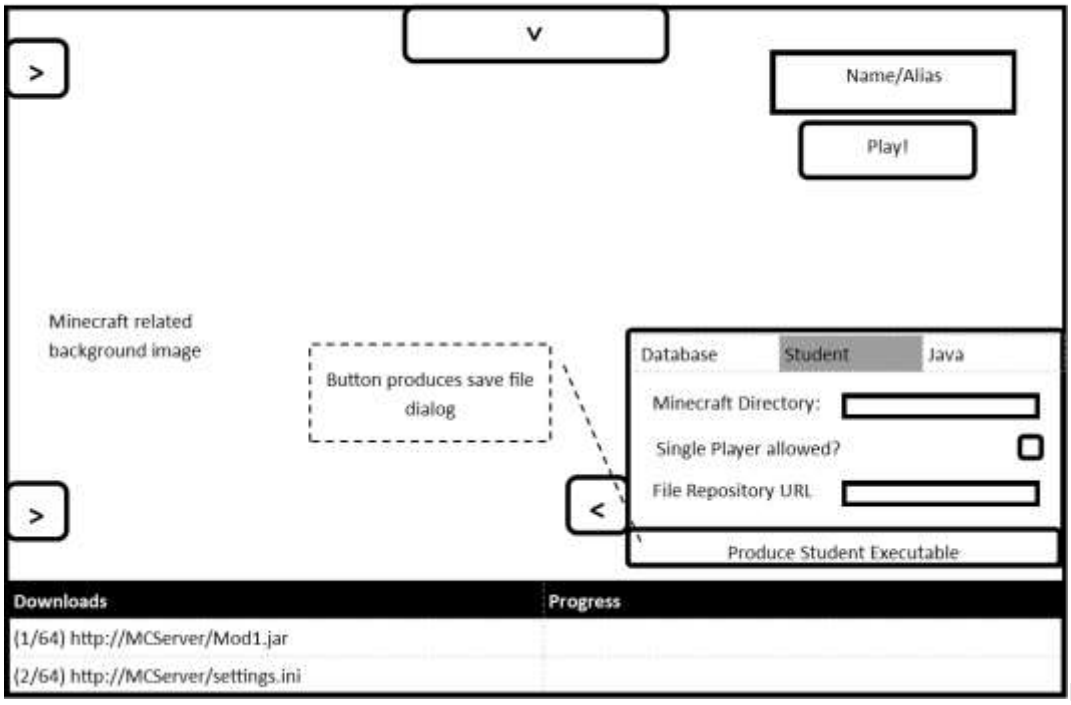
Settings Pane—Database Options



The Enable Database support checkbox disables or enables all the fields. The apply changes button should attempt to connect to the database, while the initilase database should create a new one.

Main Screen (Teacher POV)

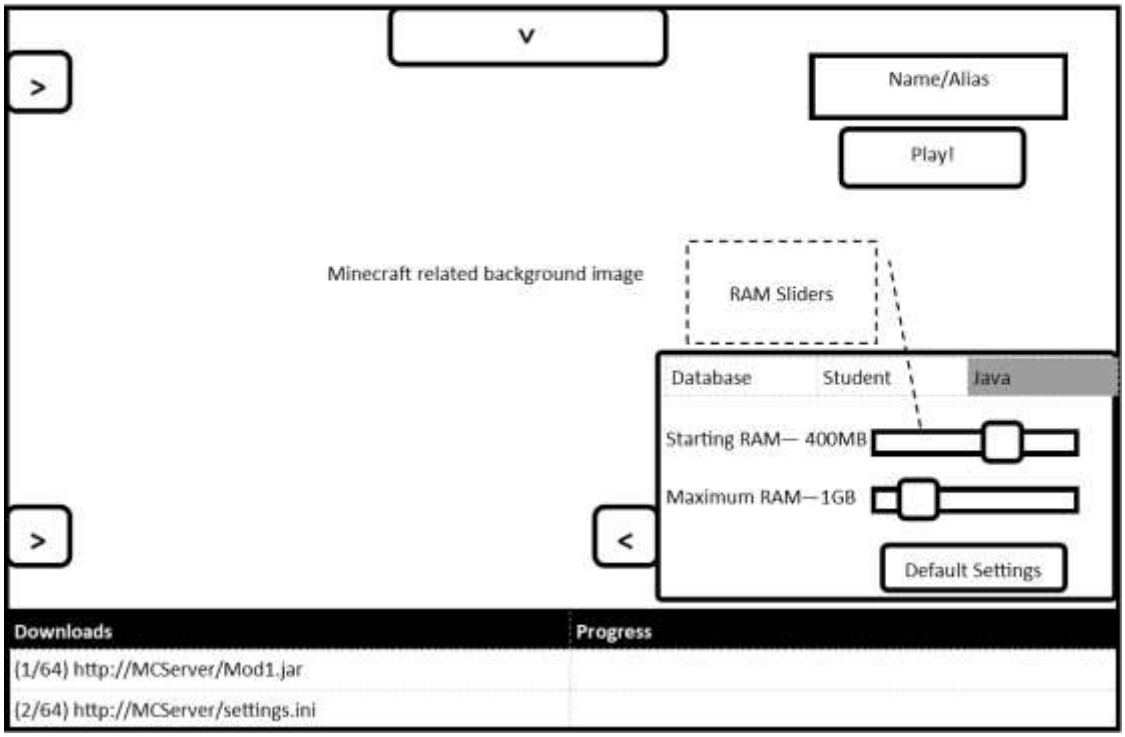
Settings Pane— Student Settings



Pressing the produce student executable button will open a save dialog. A launcher with embedded settings, for student use, will be saved to that location.

Main Screen (Teacher POV)

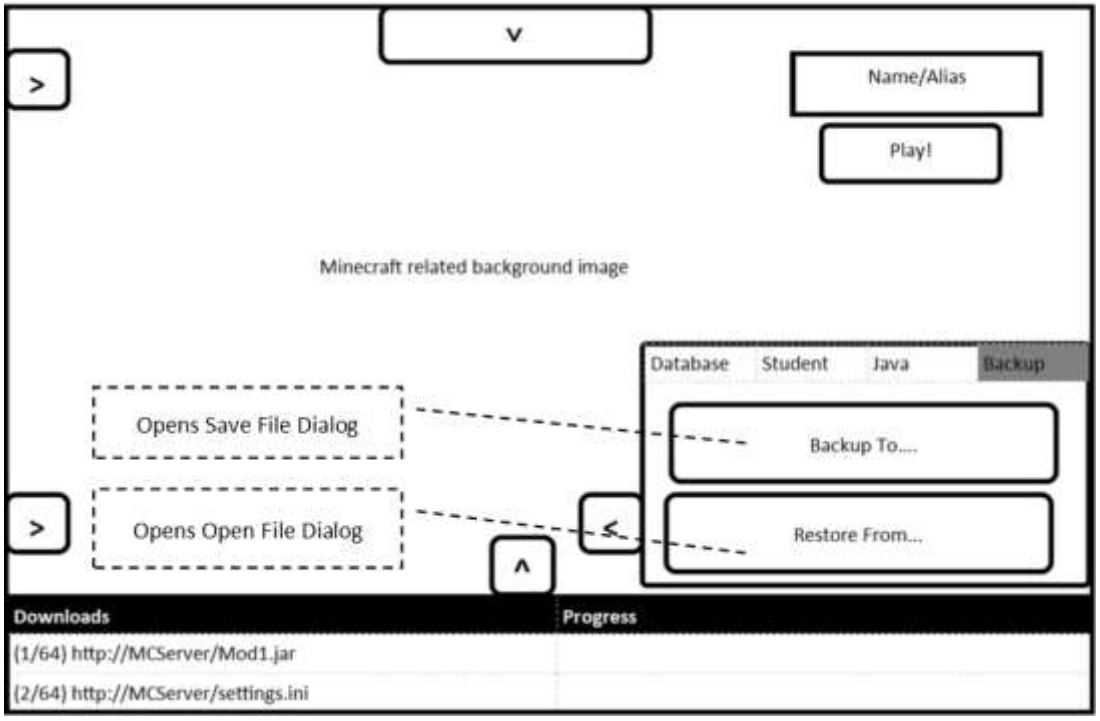
Settings Pane—Java Configuration



The JVM's RAM parameters can be configured by dragging the sliders.

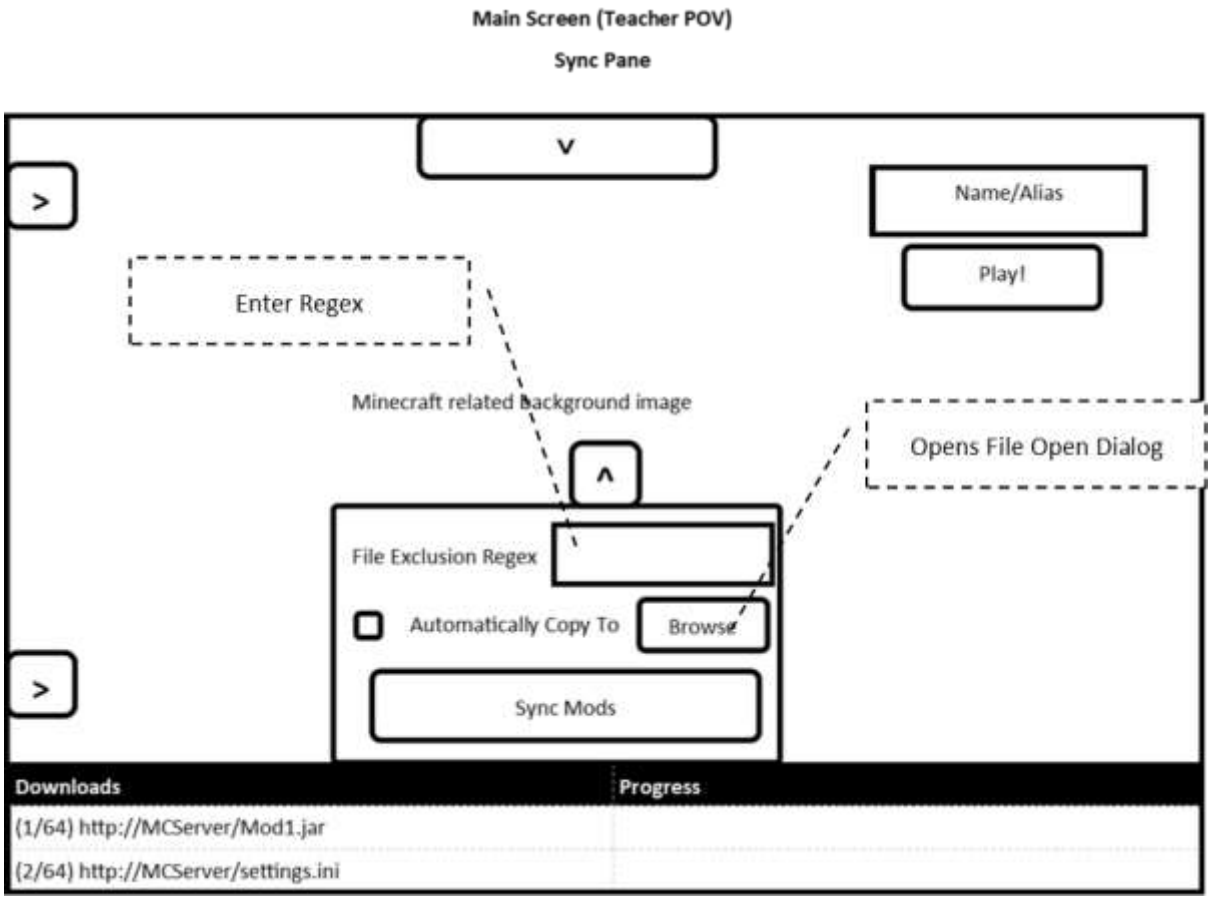
Main Screen (Teacher POV)

Settings Pane—Backup Configuration



The Backup To button will open file save dialog, and save a backup of the contents of the Minecraft directory in a ZIP file in this location. Conversely, the Restore From button restores a ZIP file.

Sync Pane



The file exclusion regex allows the teacher to enter a regular expression, whereby matching files in the teacher's Minecraft directory would not be matched.

Ticking the automatically copy to box/browse button will open up a file dialog, allowing the teacher to navigate to a directory to copy the contents of their Minecraft directory to.

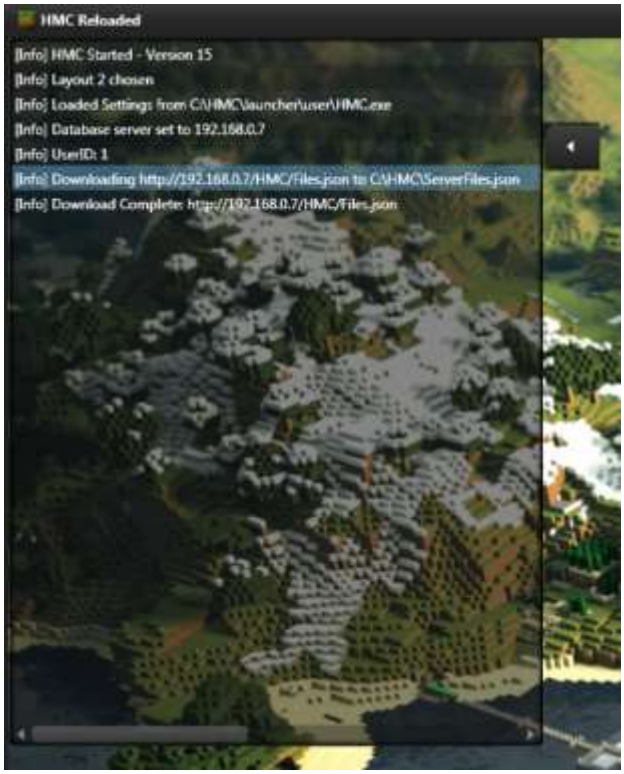
The sync mods button will apply the regex filter to the contents of the teacher's Minecraft directory, and (if selected) will automatically copy the files to the previously selected folder.

Testing

System Testing

Pupil Use:

Test 1



As expected, according to the log pane.

Test 2

As expected.

Test 3

As expected.

Test 4

As expected, with quota at top.

Test 5

After pressing quit, the launcher exited and the entry appears in the database pane under administrator mode. Success.

Test 6

Closing instantly terminates all downloads, which are re-initiated on next start up. As expected.

Test 7

Before

After

The overlay has changed, and Minecraft successfully terminates after the countdown.

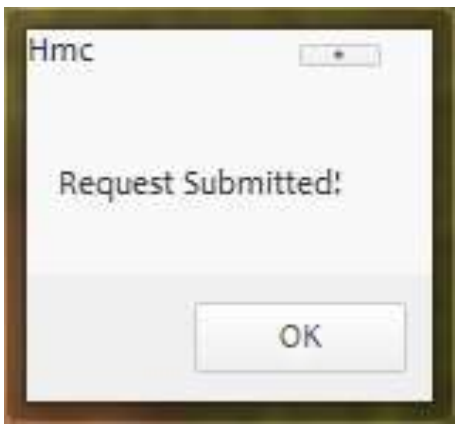
Test 8

The alias is automatically displayed in the edit box as expected.

Test 9

Before

After



Test 10

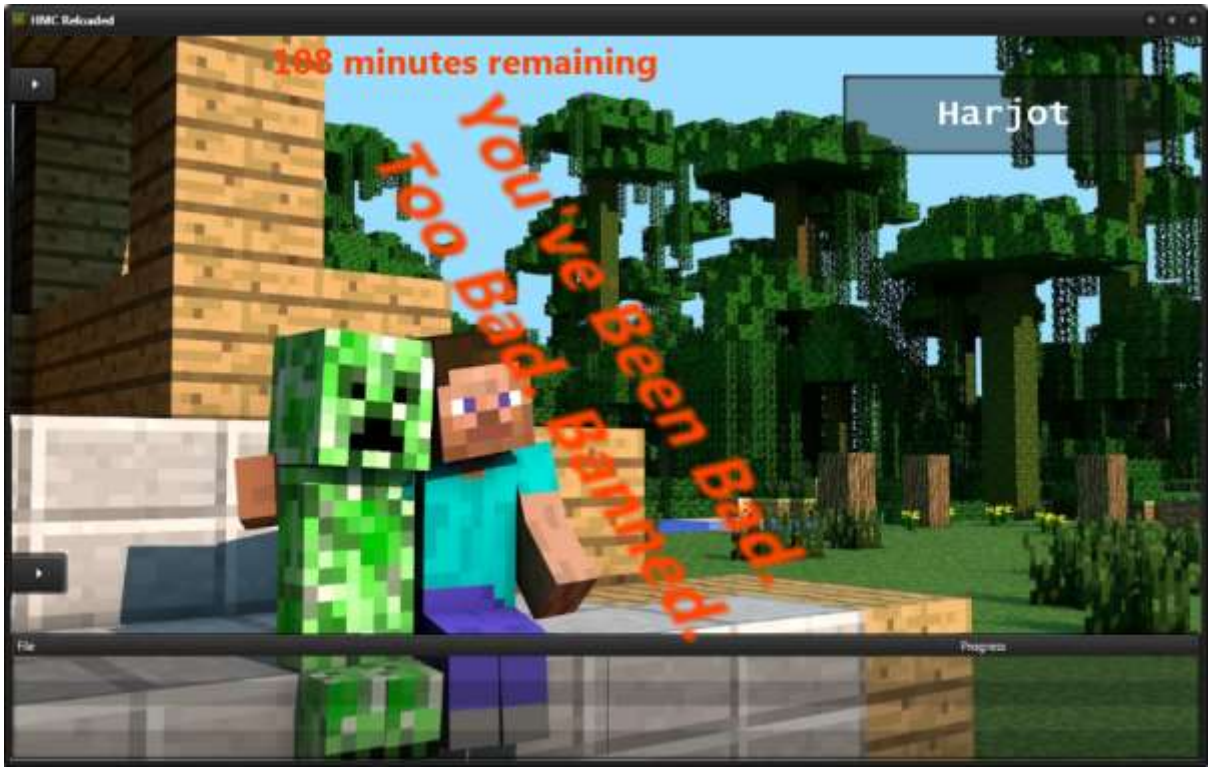
Before

After

Test 11

Before

After



Test 12

In-Minecraft

Log Pane

The log pane shows that the messages have been inserted into the database.

Teacher Use:

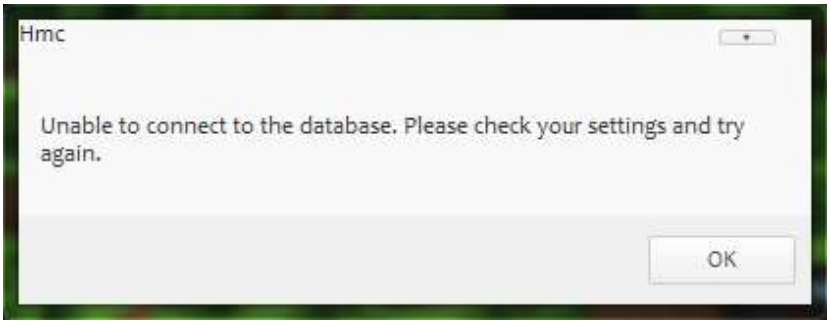
Test 1

As intended. The log pane, sync pane, Minecraft update pane, and settings pane are all available.

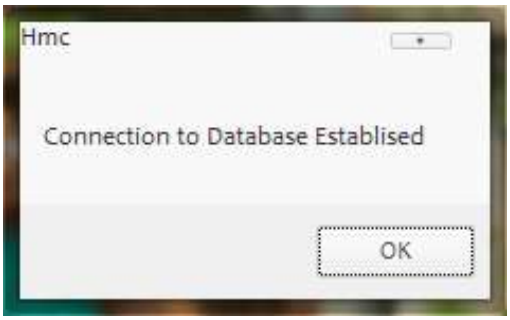
Test 2

Before:

Invalid details:



Correct details:



As intended. The details error is handled correctly with an error message, and when rectified, establishes a connection successfully.

Test 3

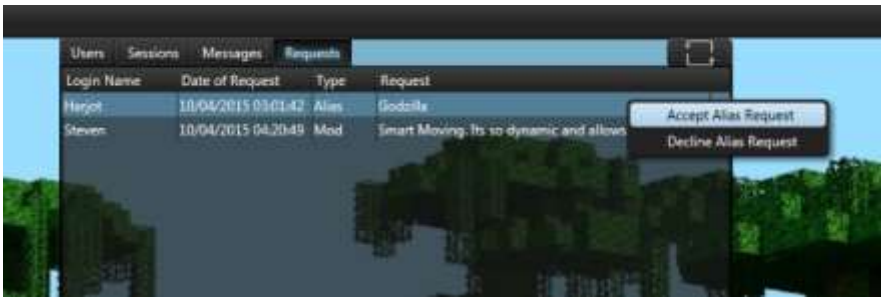
Before:

Dialog:

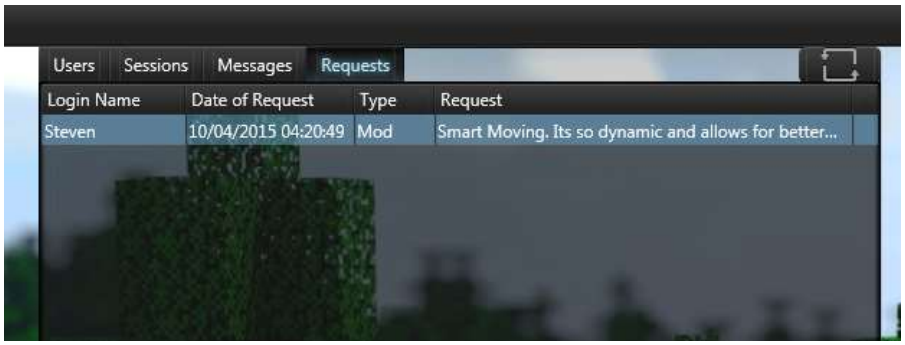
The database is restored to its initial state, and the current user is inserted straight away as an administrator as intended.

Test 4

Before:



After:



The alias has clearly been updated, as intended:

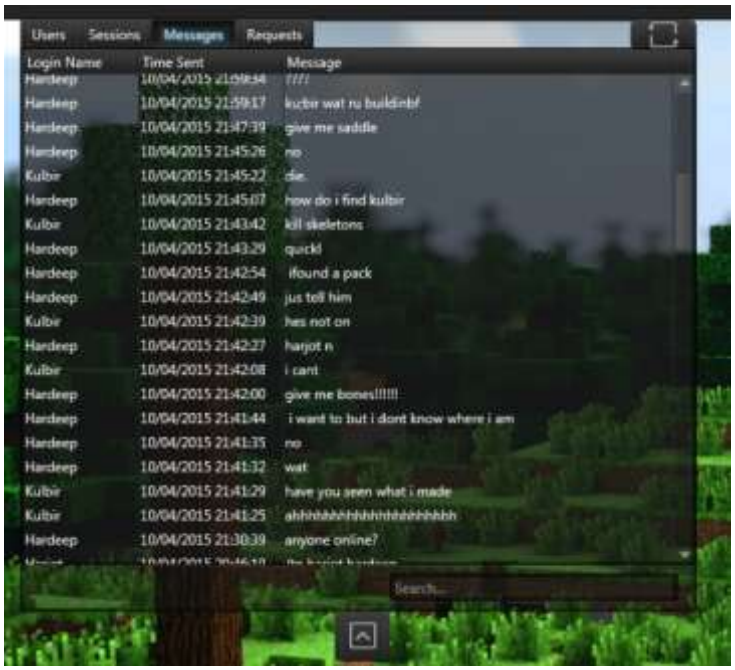
Test 5

Before:

After:

The mod request has been successfully hidden from the log pane, as expected.

Test 6



User messages are displayed in the database pane as expected.

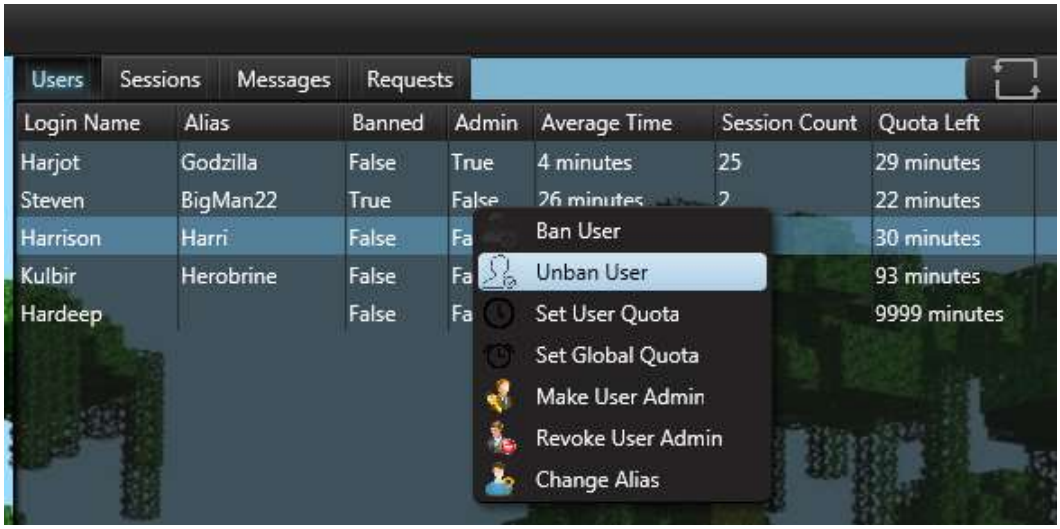
Test 7

Ban Before:

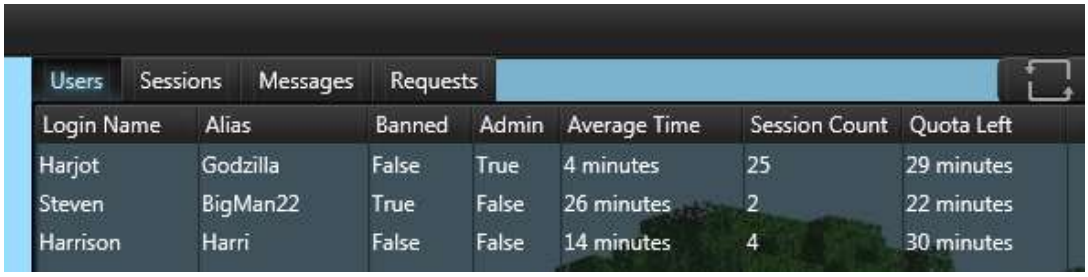
Ban After:

The user has been successfully banned as expected.

Unban Before:



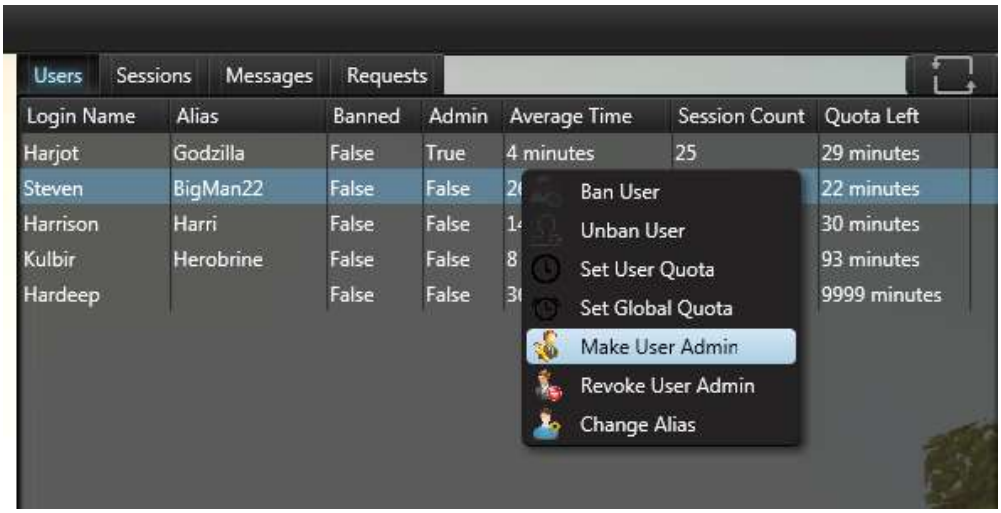
Unban After:



Banning and unbanning was performed as expected.

Test 8

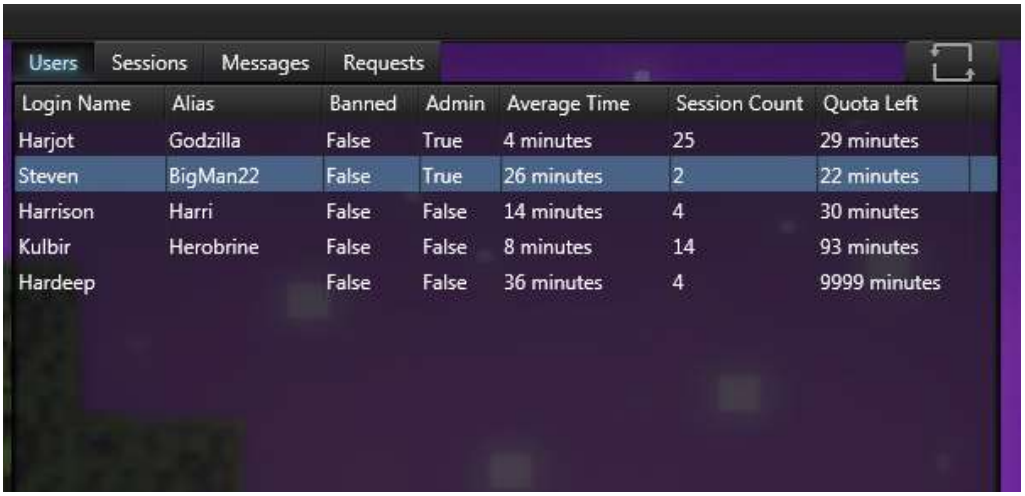
Before Assign:



After Assign:

Before Revoke:

After Revoke:

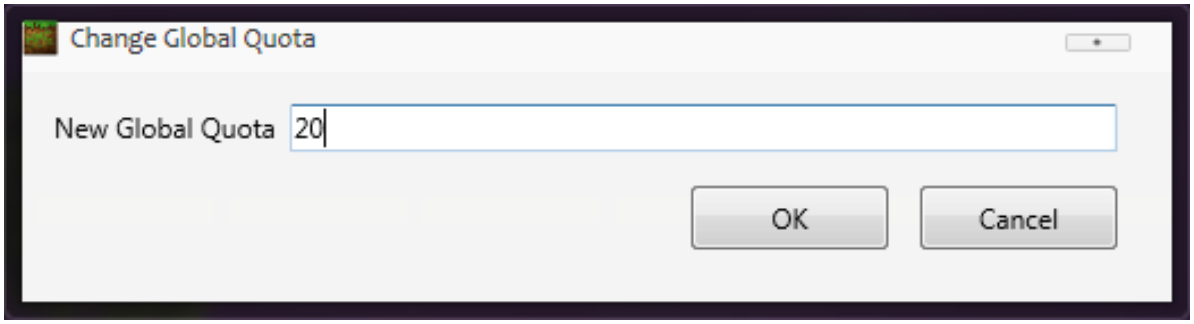


As expected. The Admin property has been succesfully assigned and revoked.

Test 9

Before:

After:



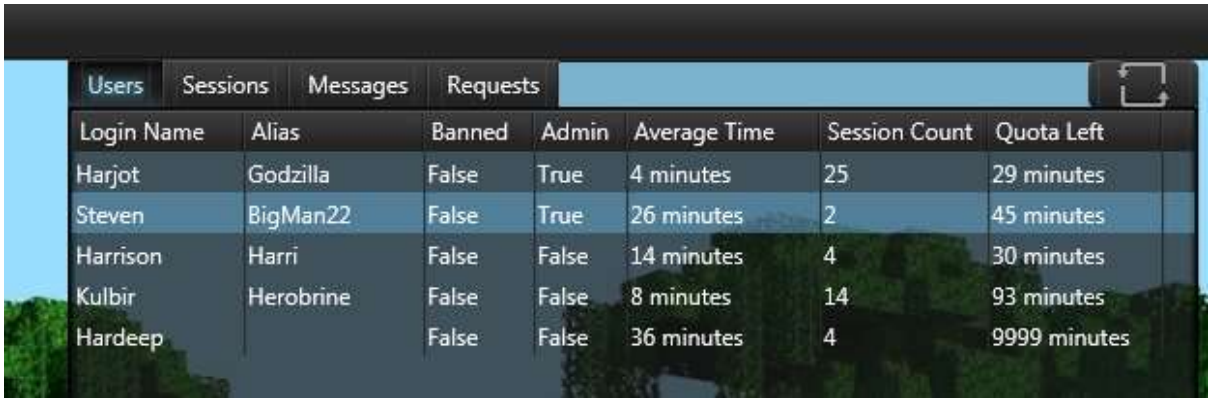
The global quota of the quota has been changed to 20, once “OK” was pressed.]

Test 10

Before:

Dialog:

After:



The Quota Left field reflects that the user quota has changed as expected.

Test 11

Before:

After:

The trailing '/' at the end of the file repository URL was removed, and the result is as expected.

Test 12

Before:

After:

Test 13

Before:

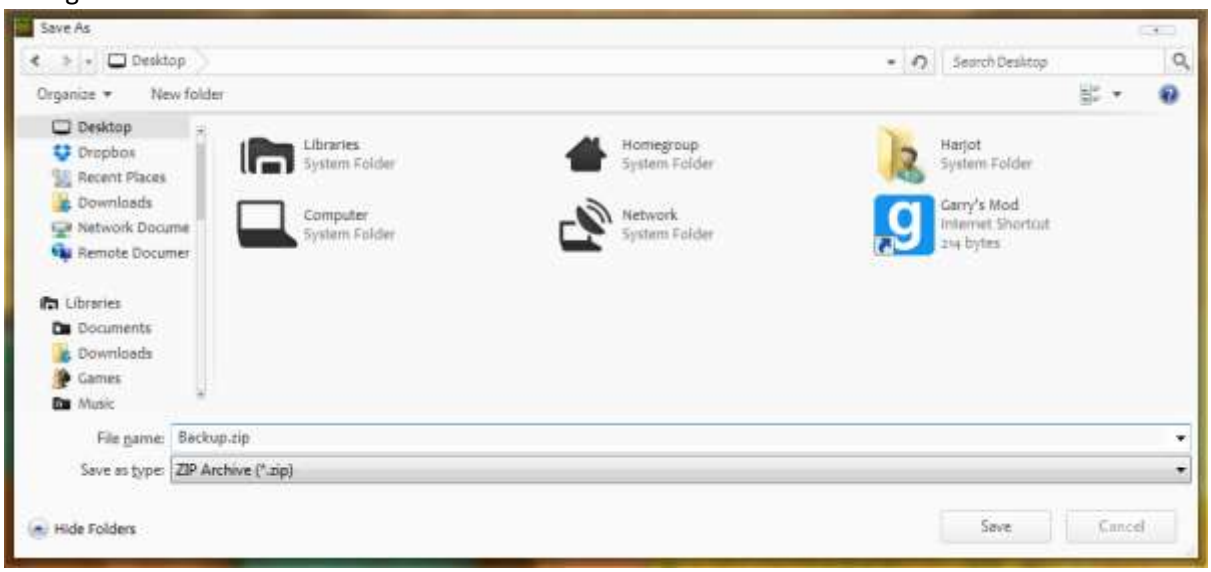
After:

The test was sucessful, and files began downloading for Minecraft 1.8.

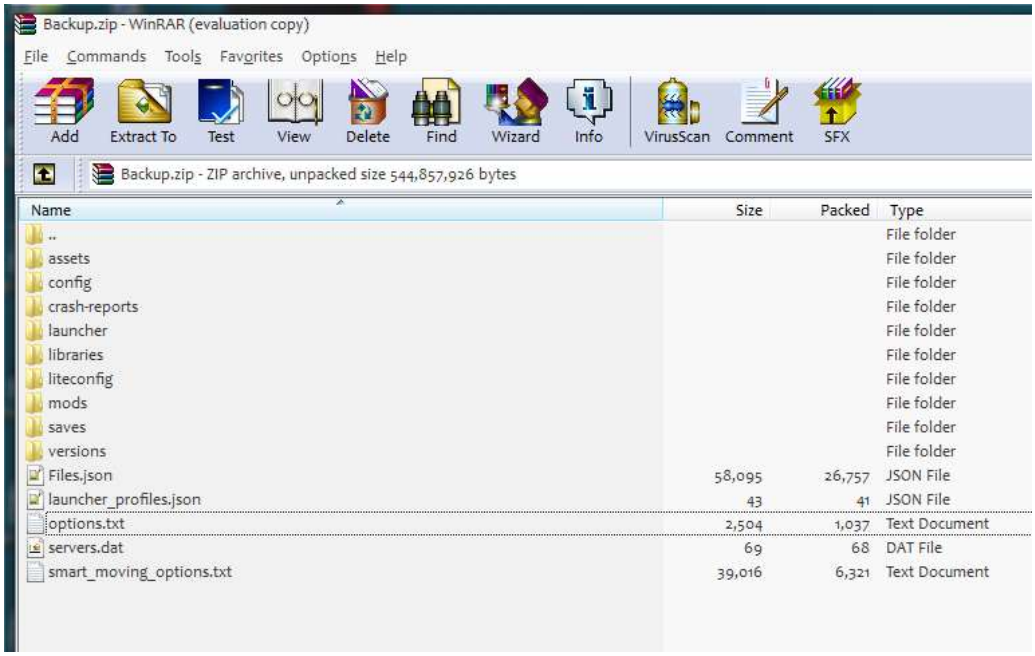
Test 14

Before:

Dialog:



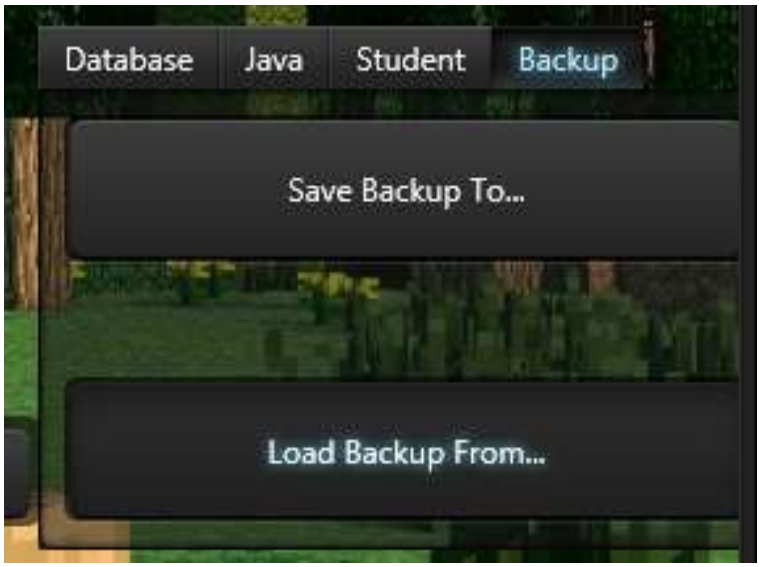
After:



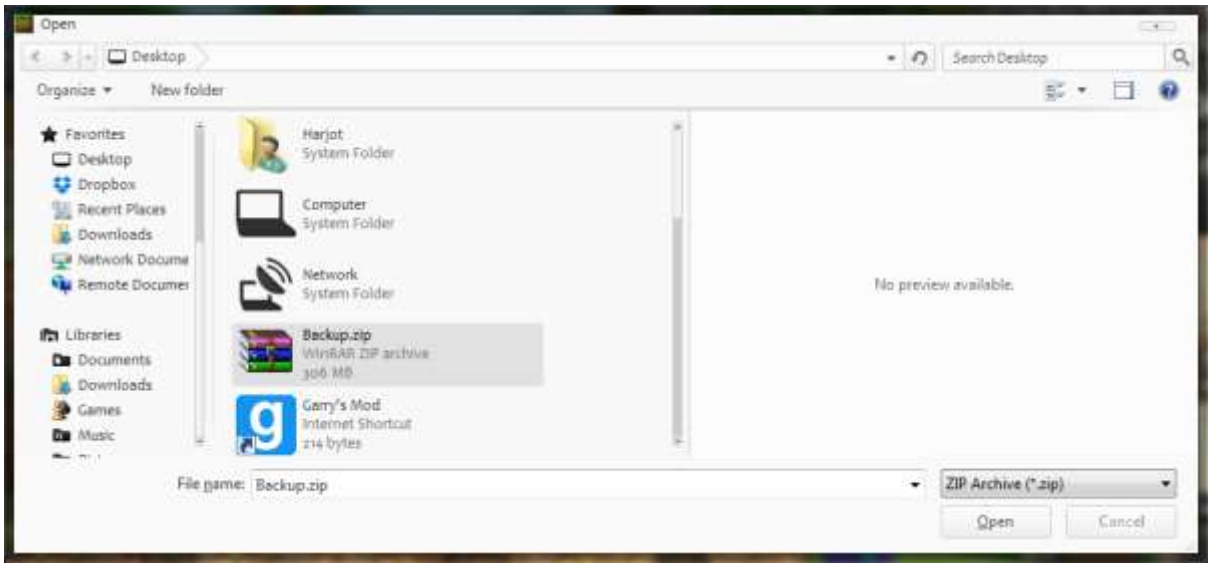
Clearly creating the backup archive was as expected, and looking at the file in WinRar proves this.

Test 15

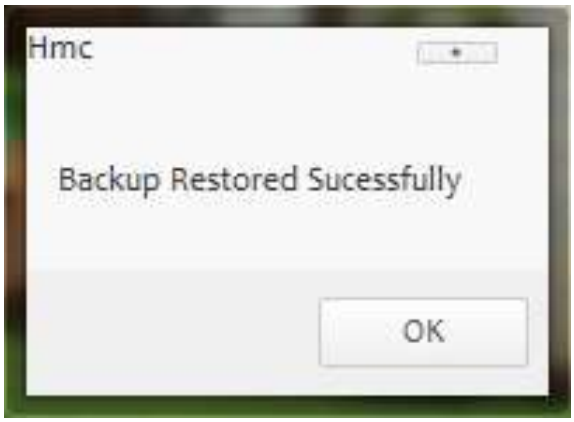
Before:



Dialog:



After:



The resulting dialog indicates the test was as expected.

Test 16

Before:

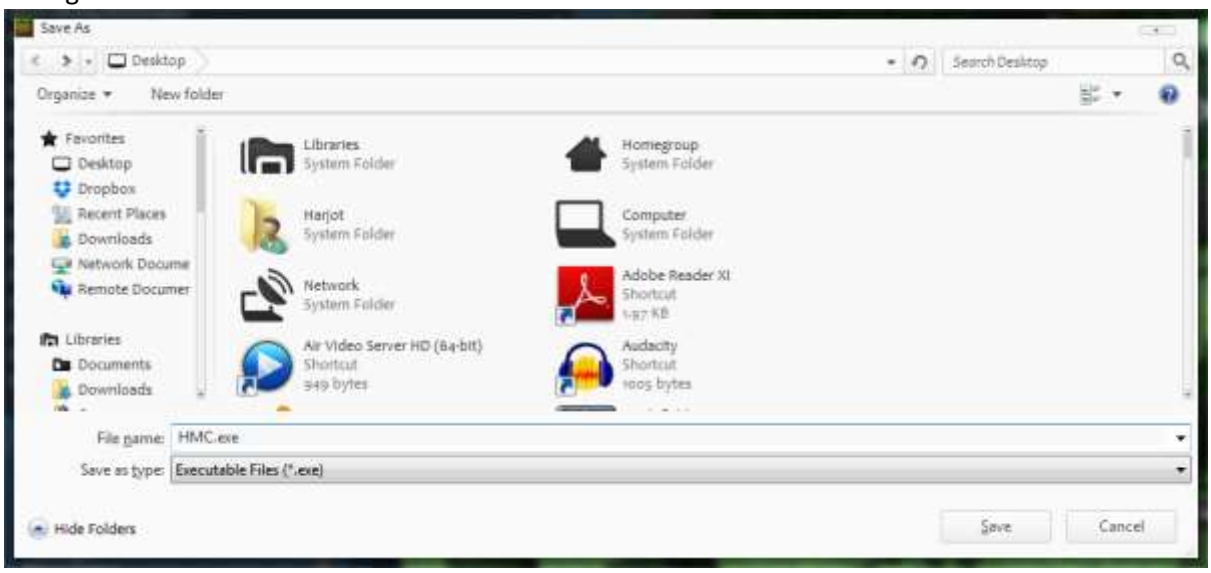
After:

The log pane shows that the file list was generated, and saved to the right directory under the right file name. Test passed.

Test 17

Before:

Dialog:

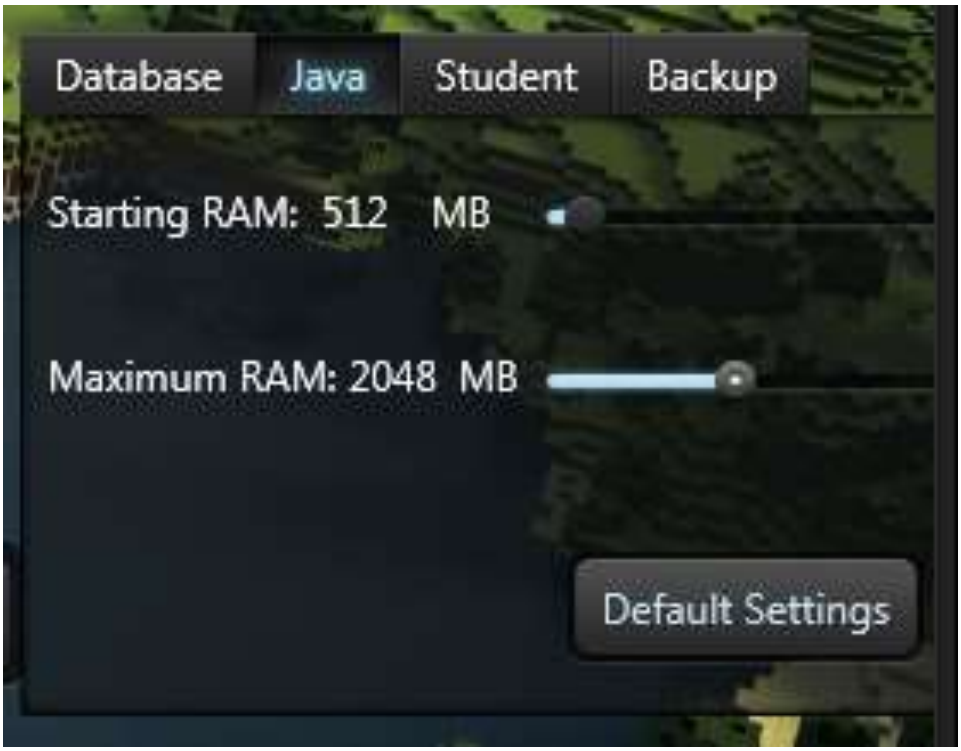


After:

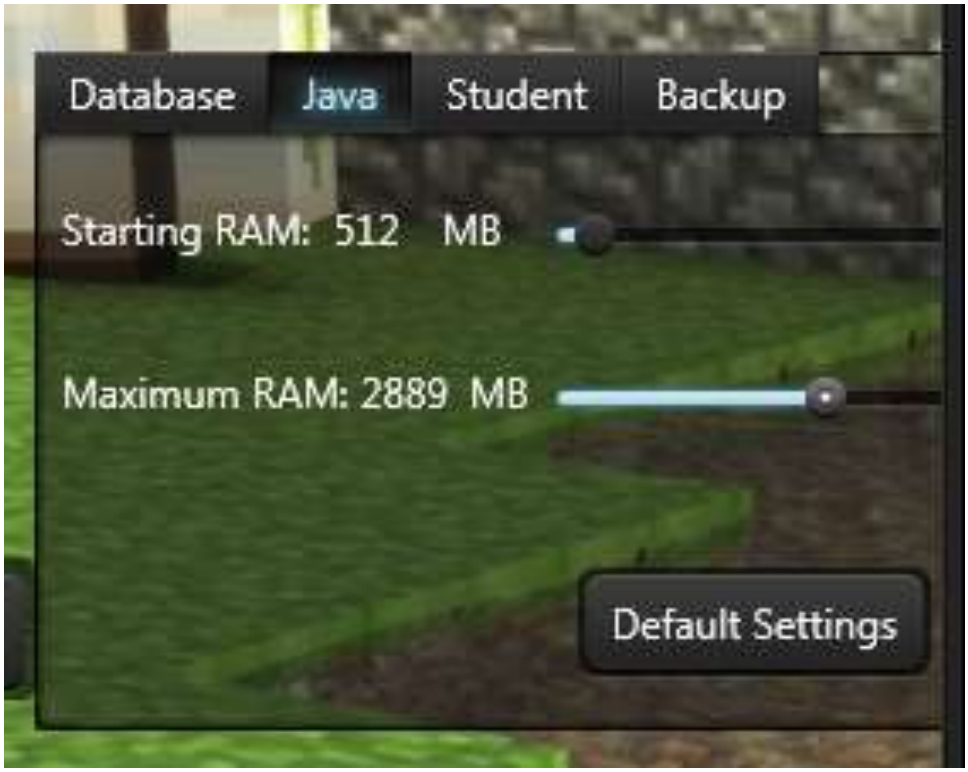
The log pane proves that the test results as expected.

Test 18

Before:



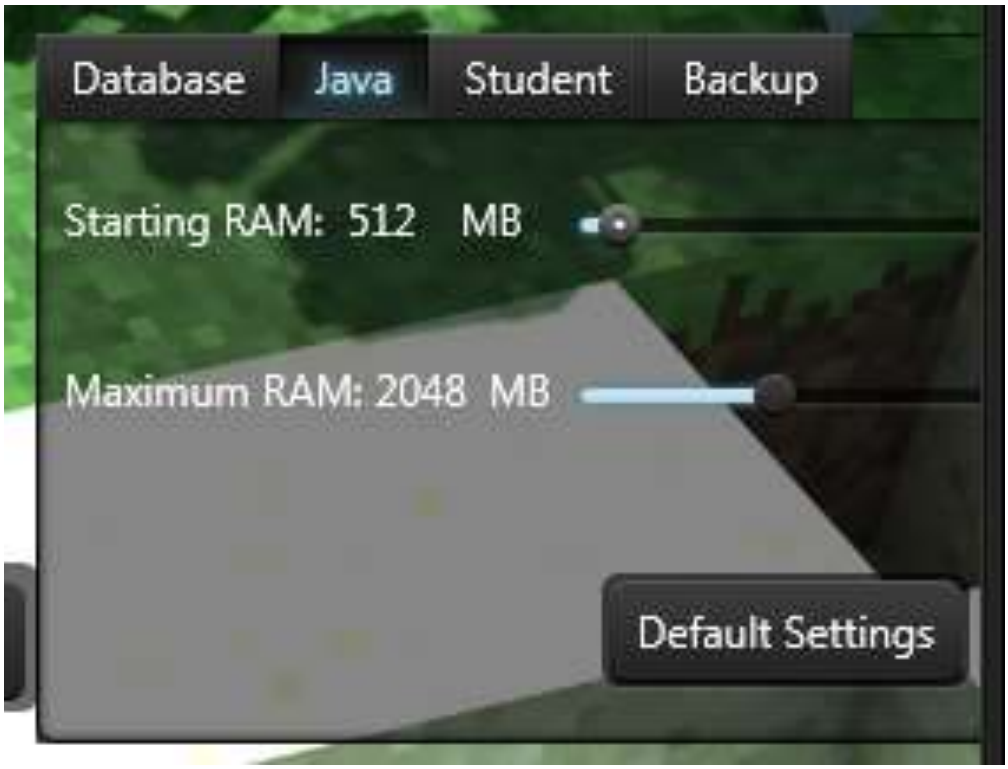
After:



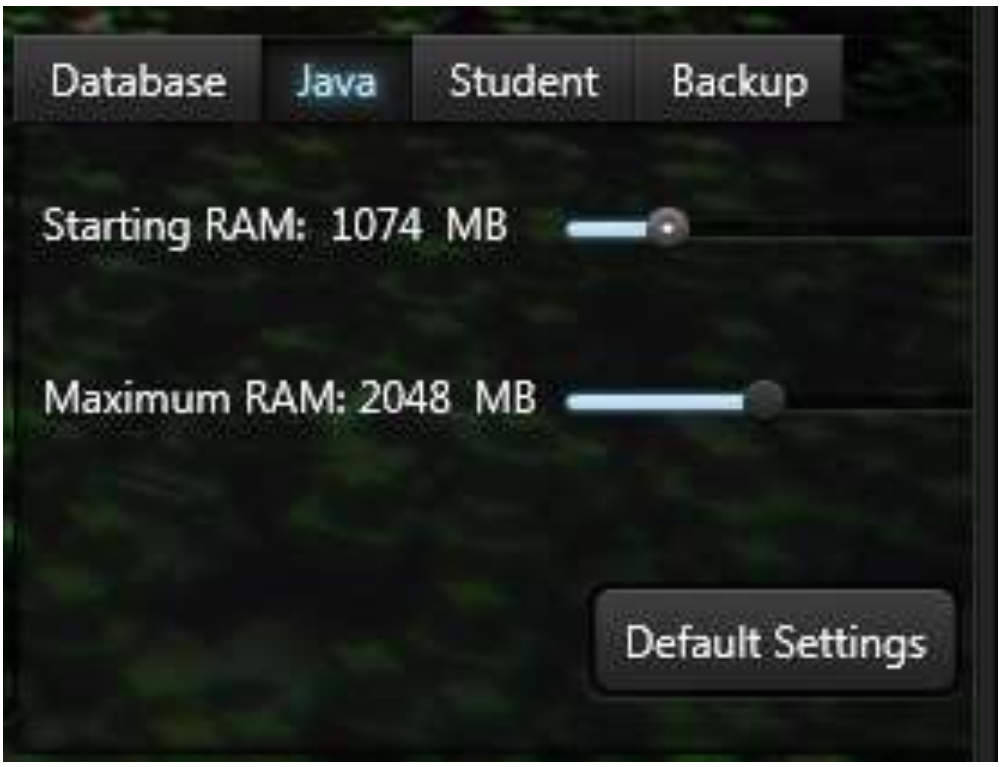
The maximum RAM slider changes as expected, and saves settings.

Test 19

Before:



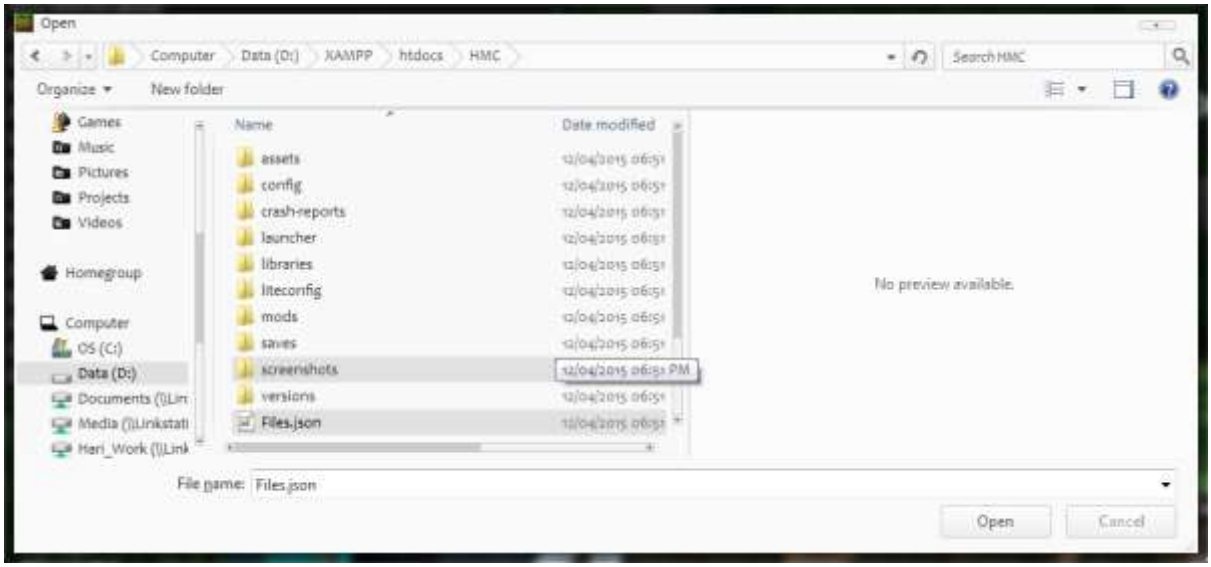
After:



Test 20

Before

Dialog

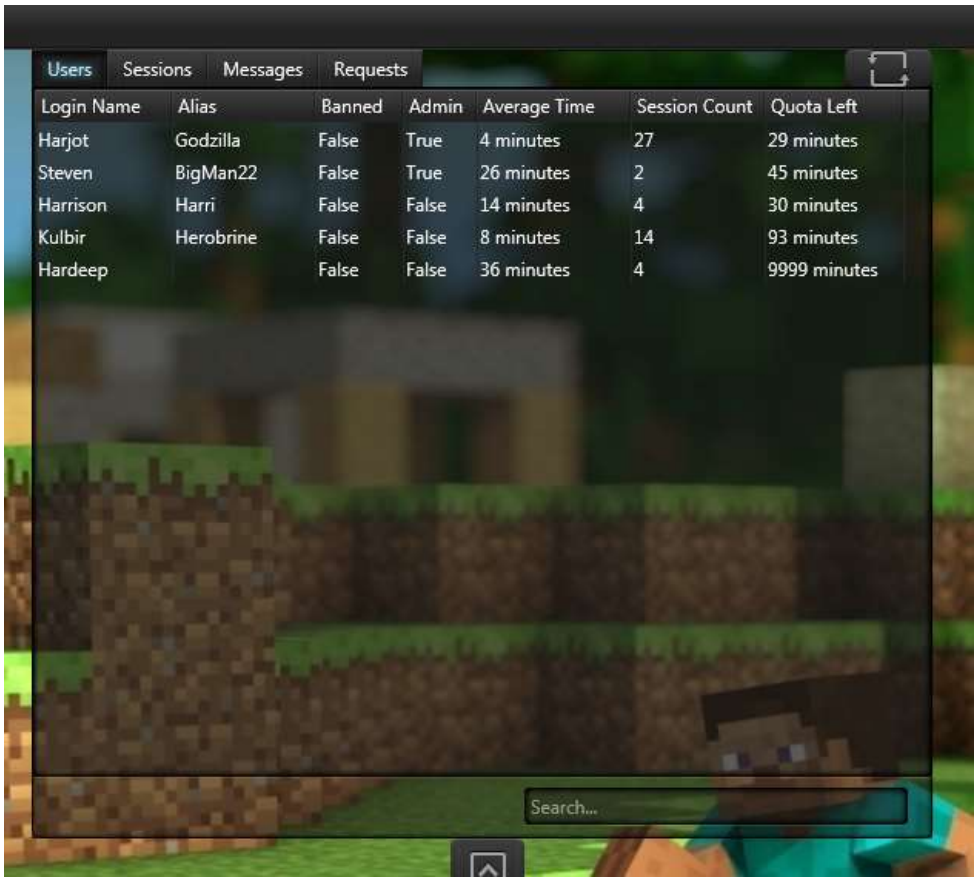


After

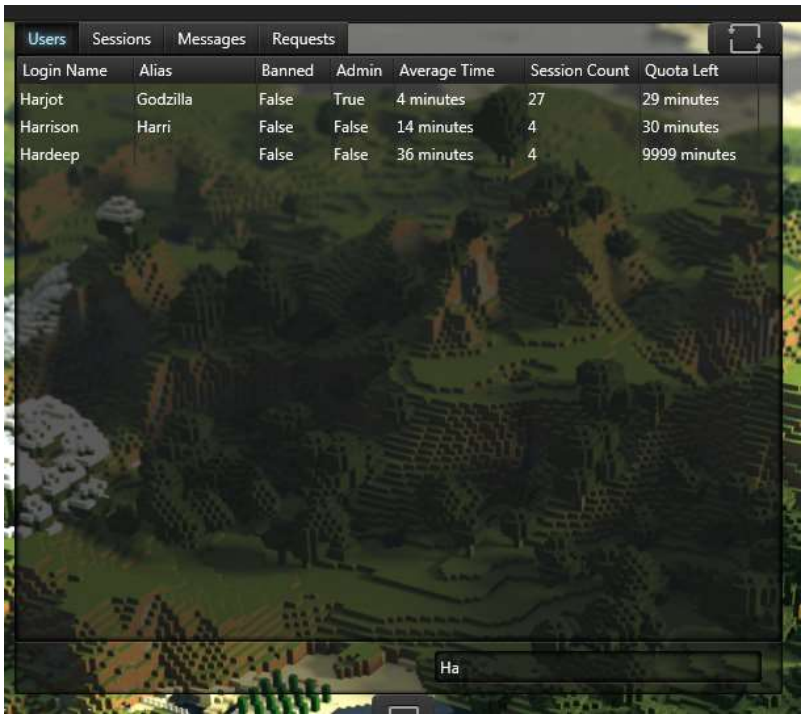
The correct path has been reflected after the checkbox. Test passed.

Test 21

Before



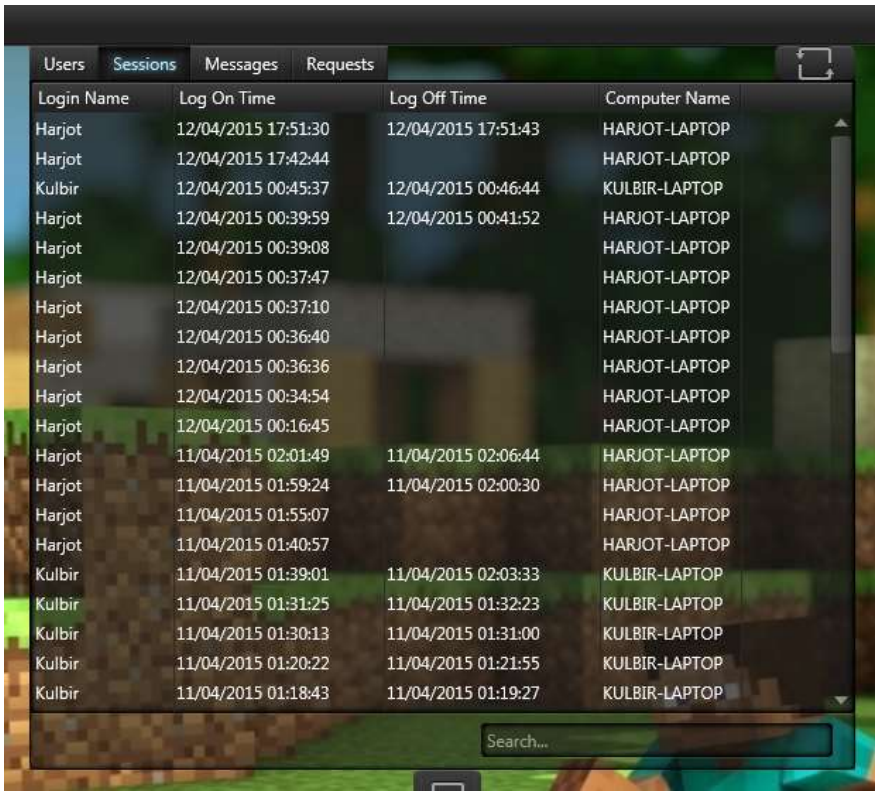
After



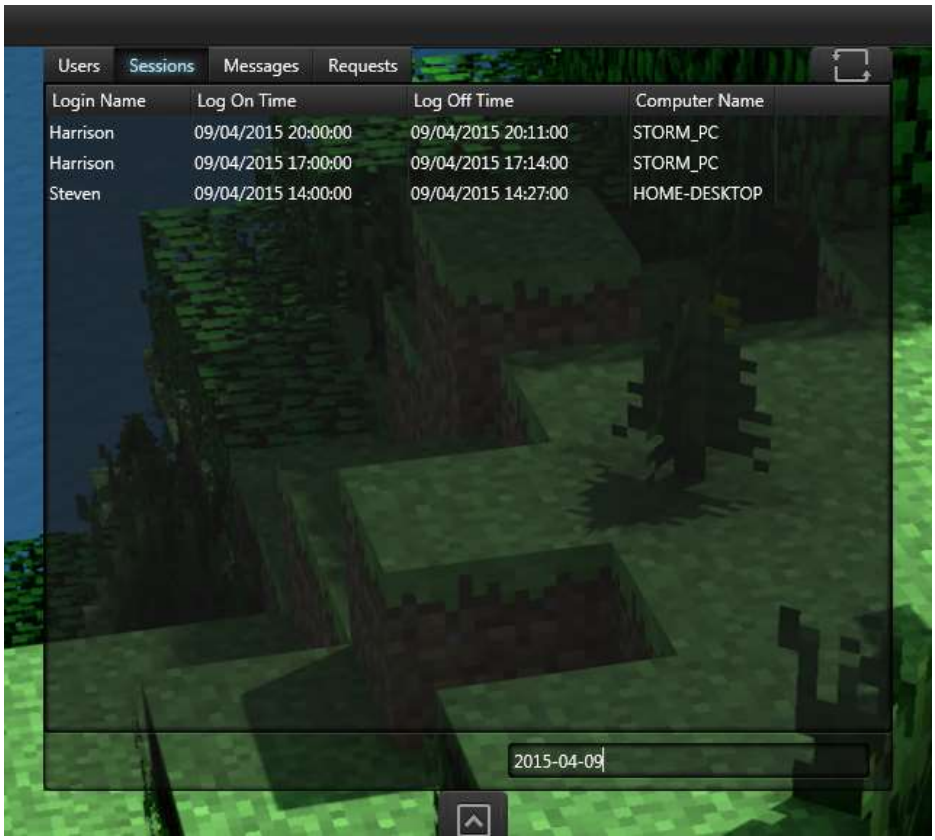
The users have been filtered out correctly with the search term “Ha”, as expected.

Test 22

Before



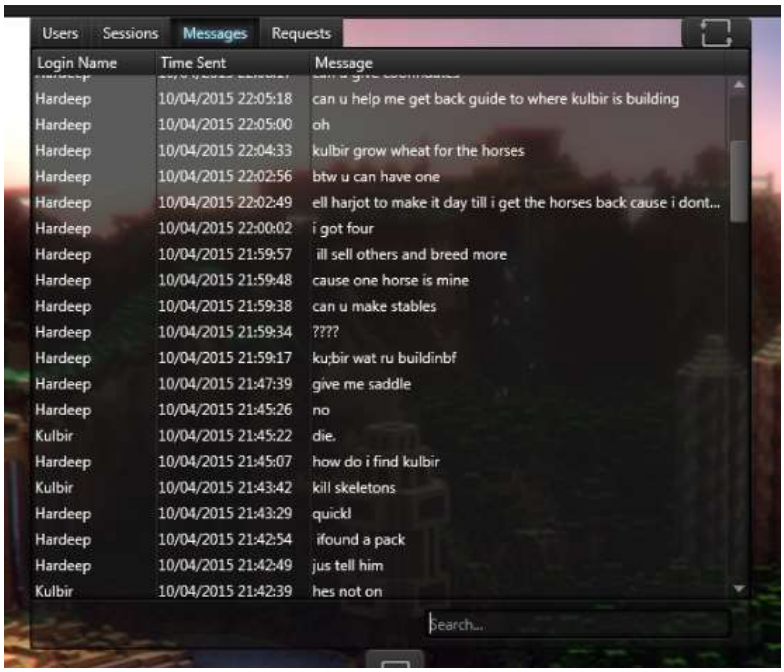
After



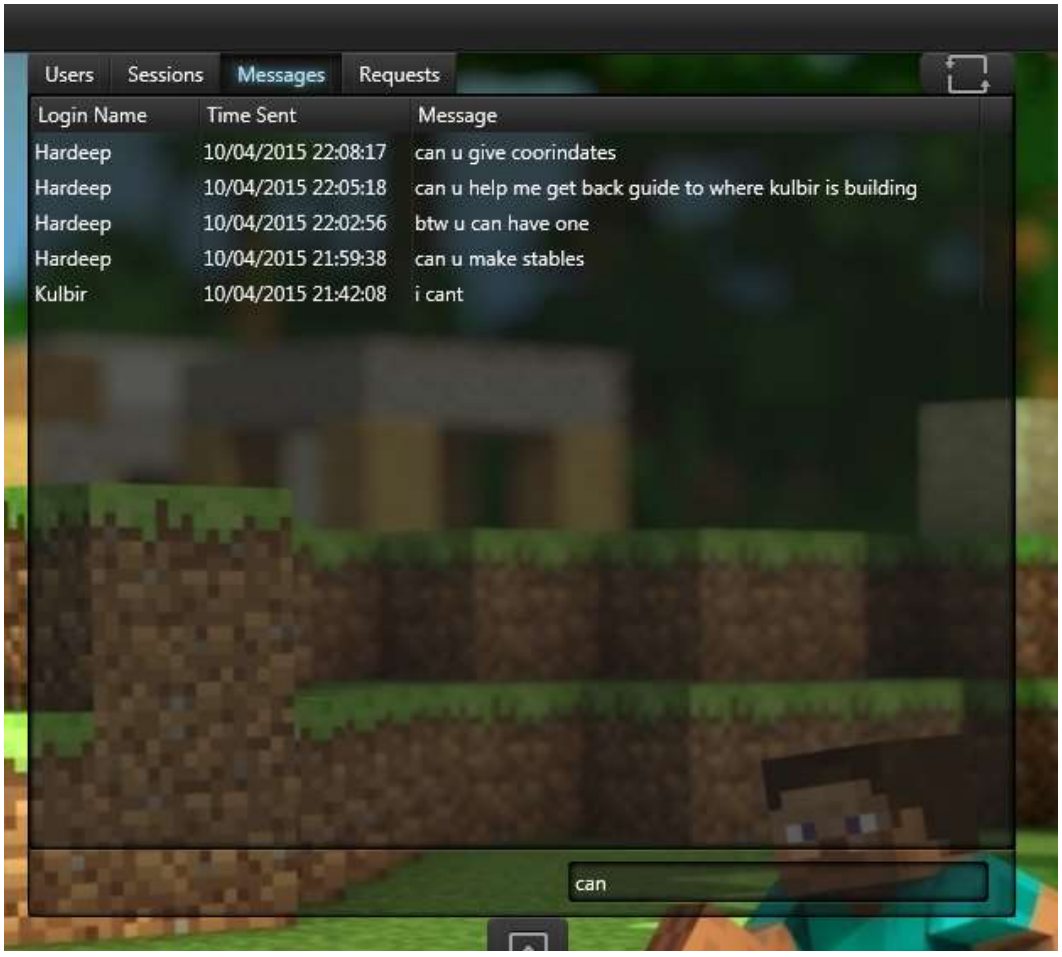
The search returns as expected, but requires any dates to be entered in the format shown above.

Test 23

Before



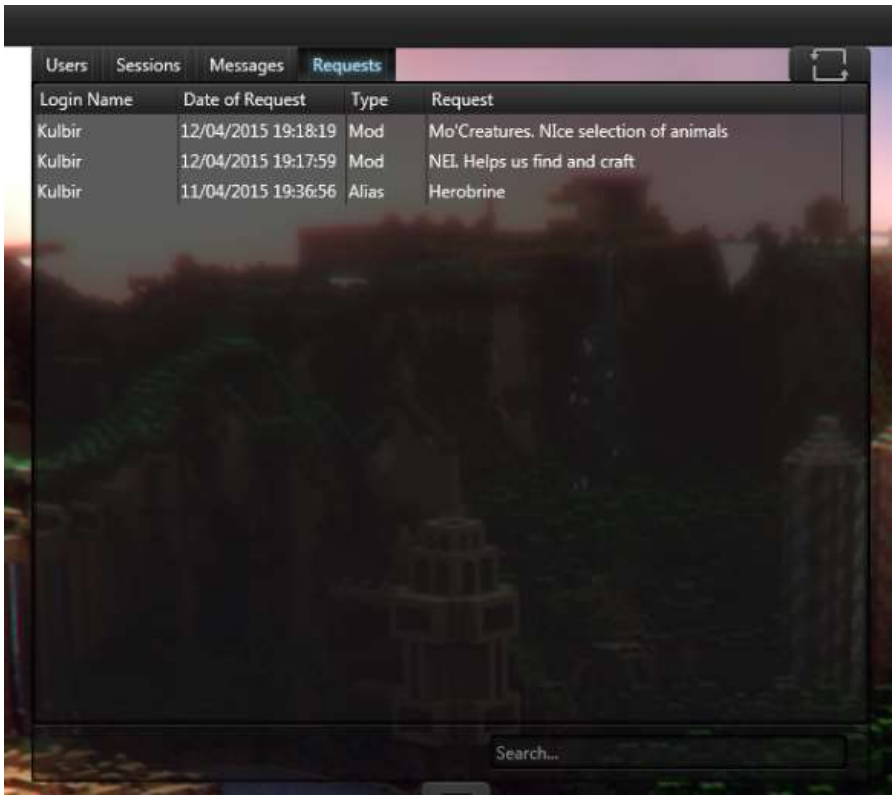
After



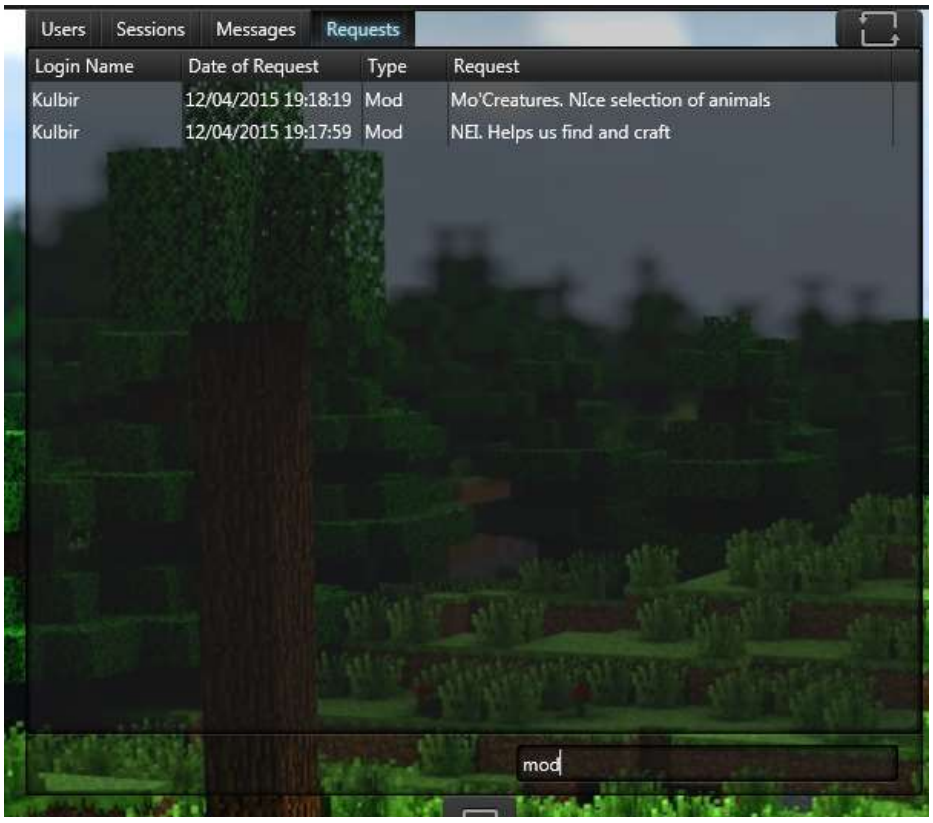
The test shows that messages matching the search term appear successfully.

Test 24

Before



After



The search bar filters results as expected.

Validation Testing

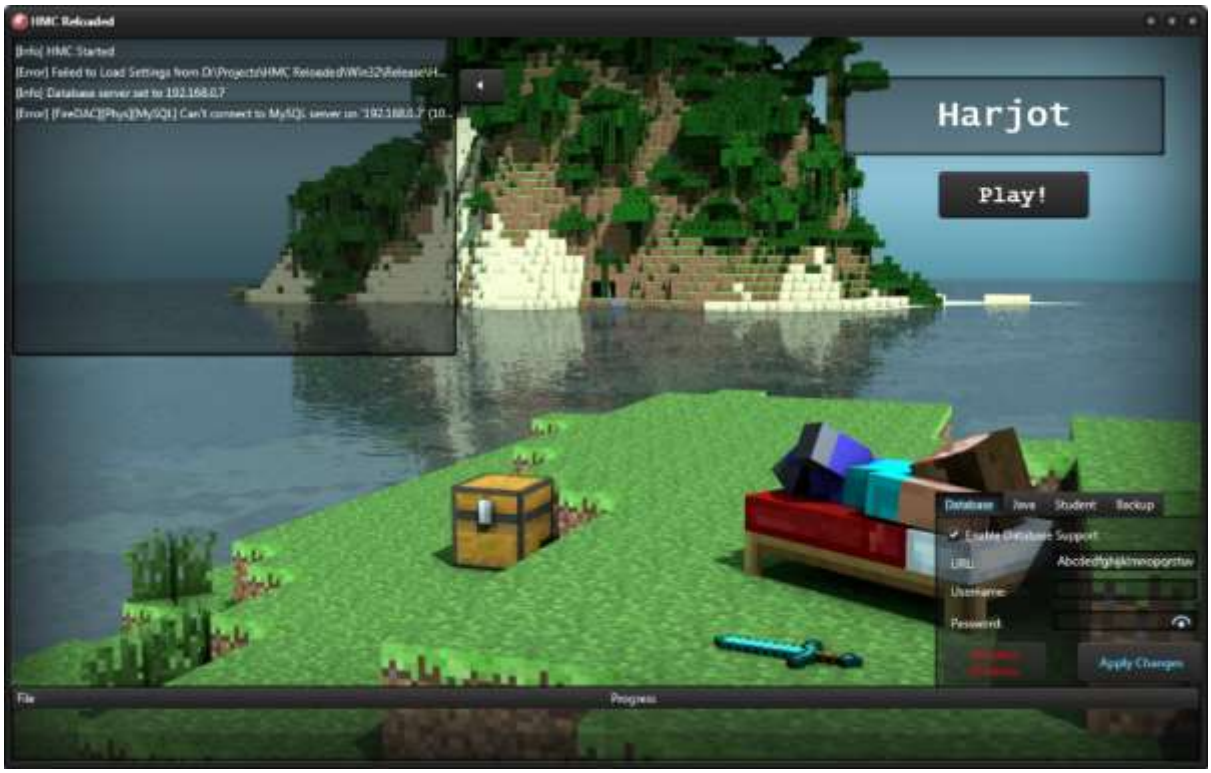
Test 1 - Pass

Before

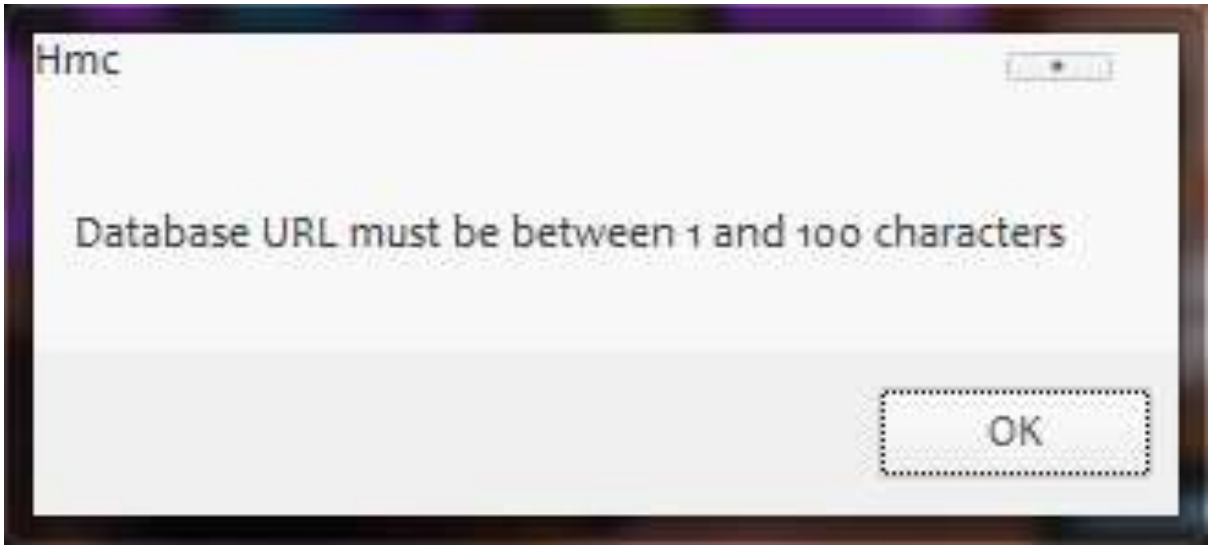
After

Test 2 - Pass

Before



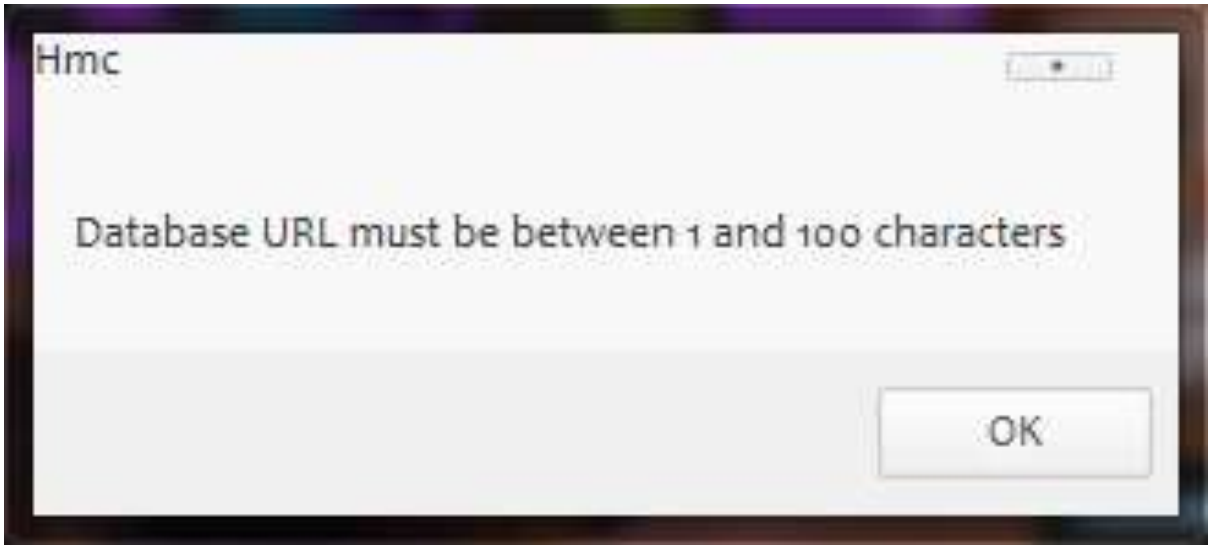
After



Test 3 - Pass

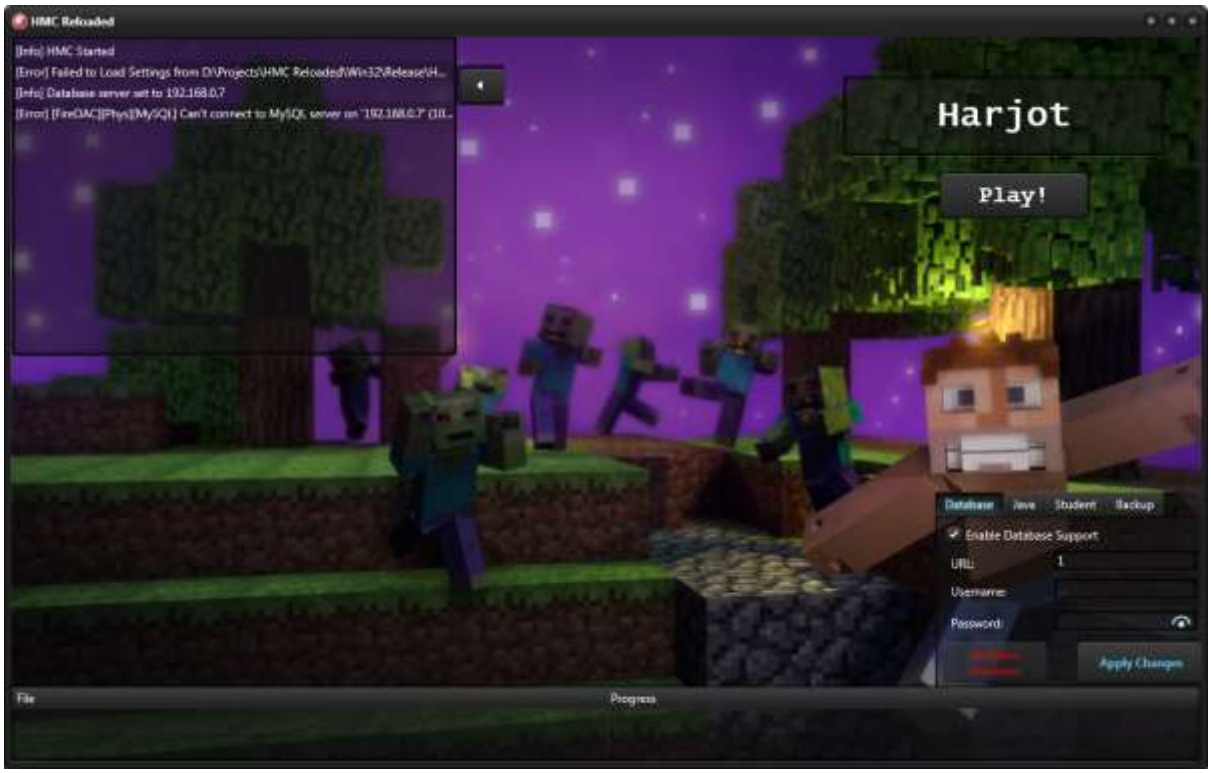
Before

After

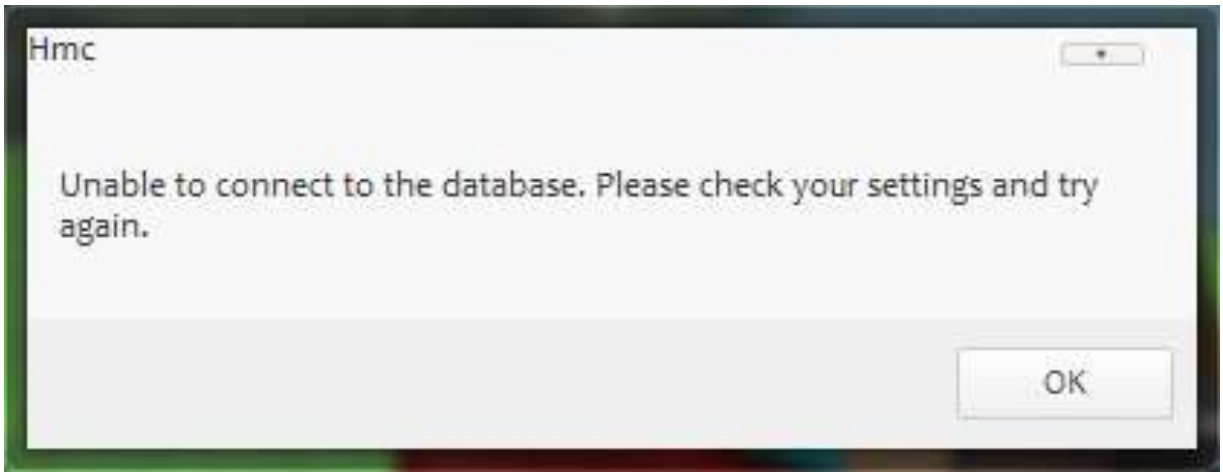


Test 4 - Pass

Before



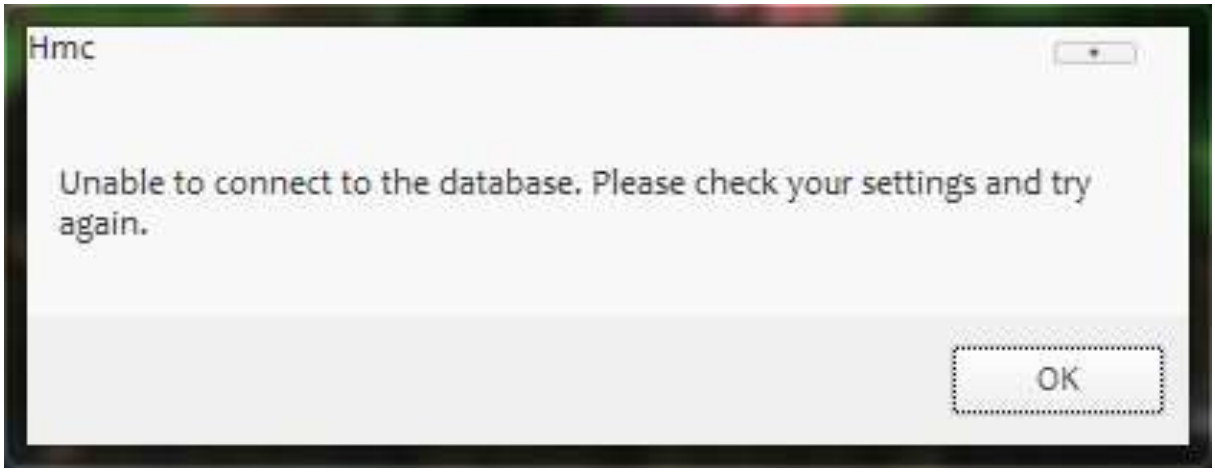
After



Test 5 - Pass

Before

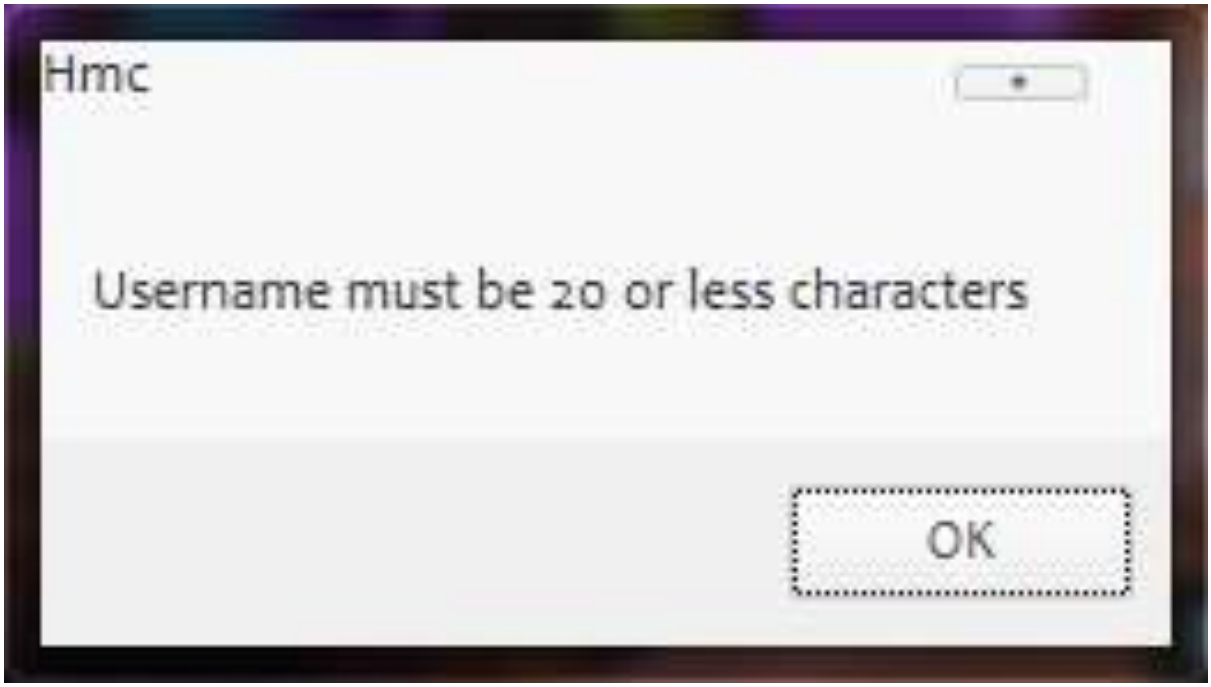
After



Test 6 - Pass

Before

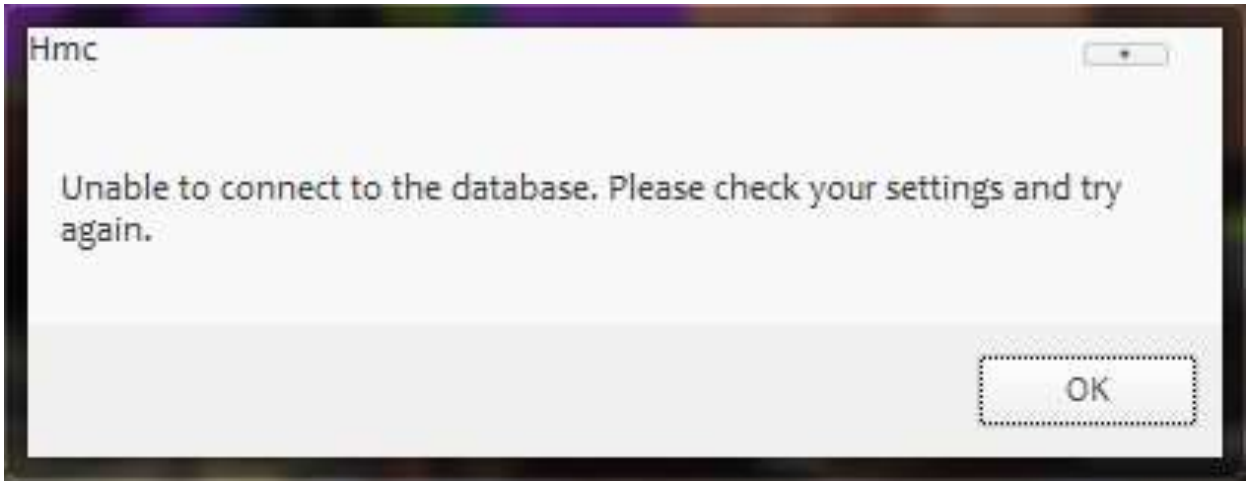
After



Test 7 - Pass

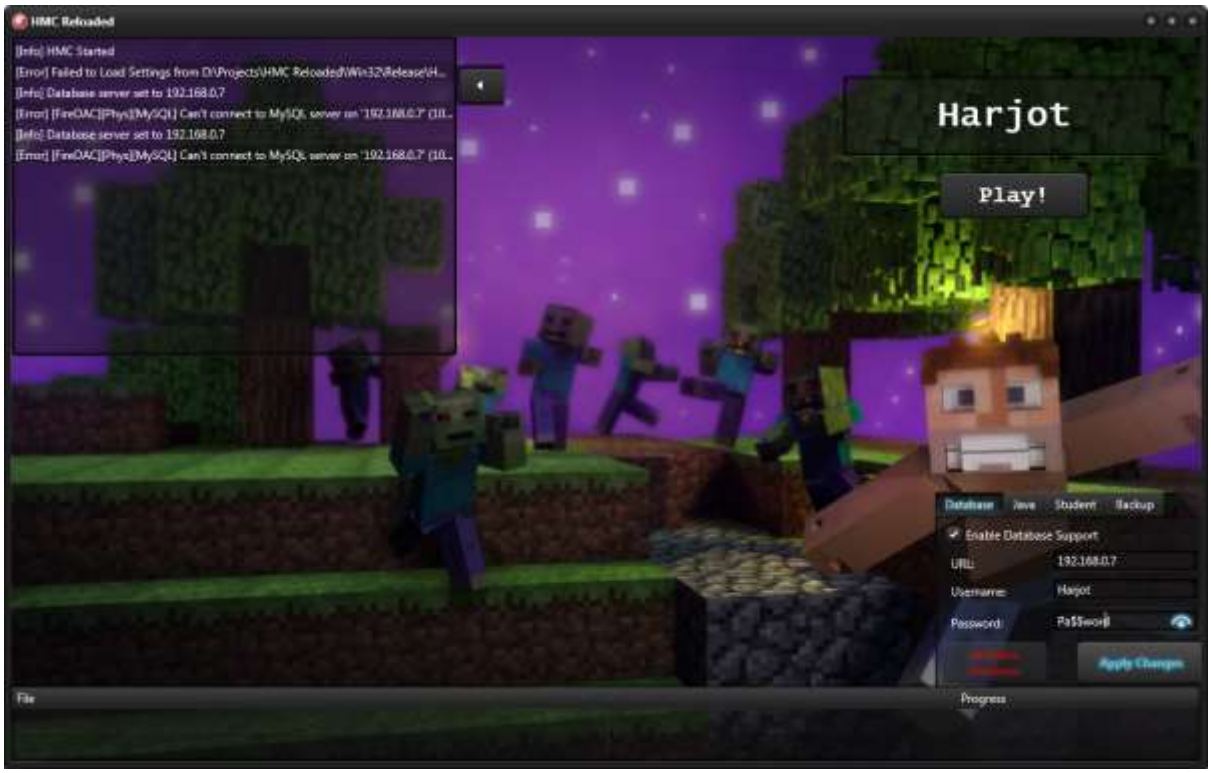
Before

After

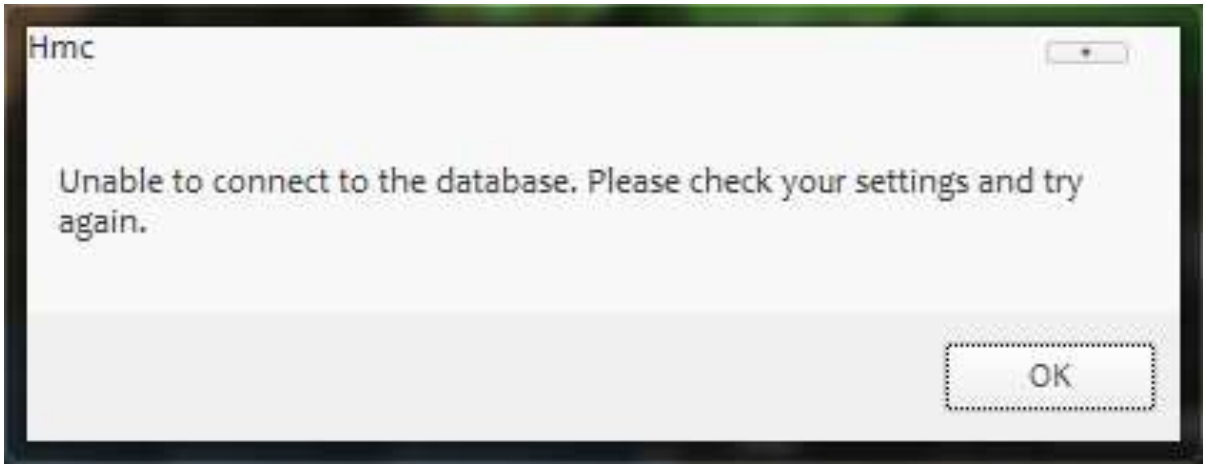


Test 8 - Pass

Before



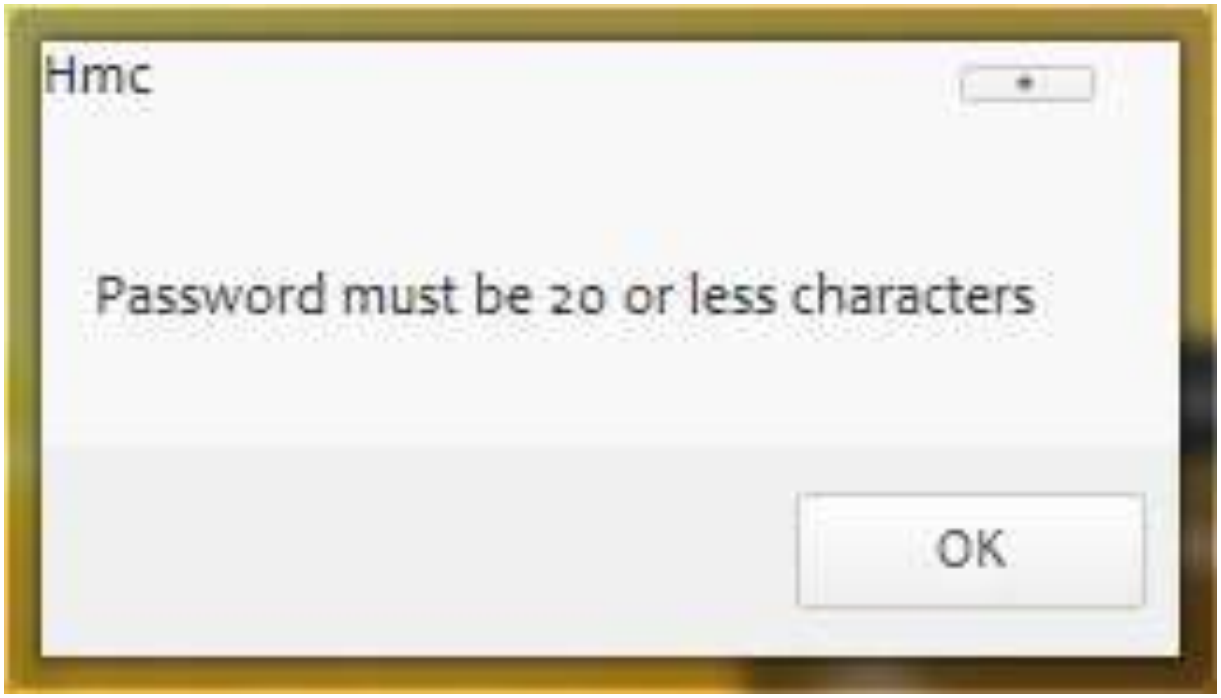
After



Test 9 - Pass

Before

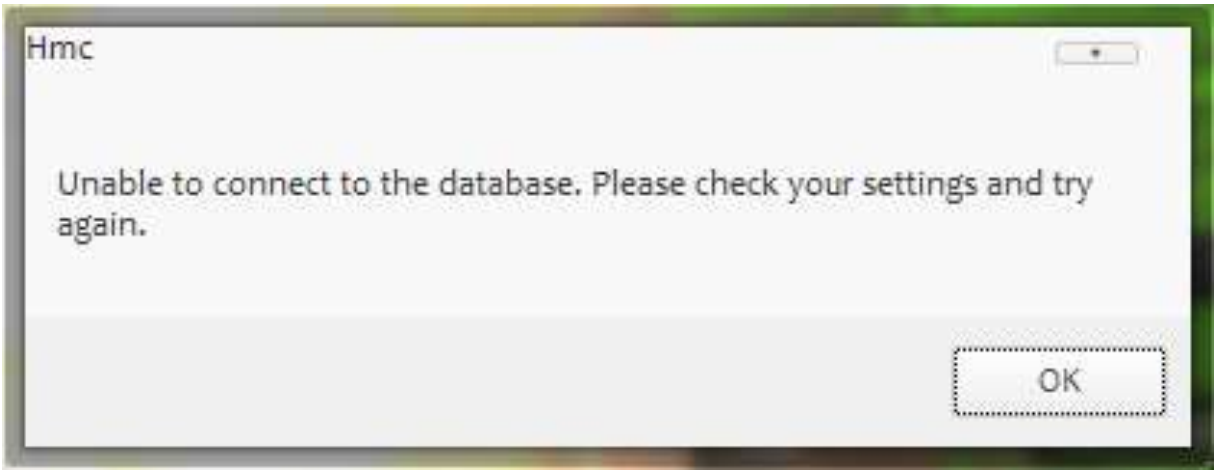
After



Test 10 - Pass

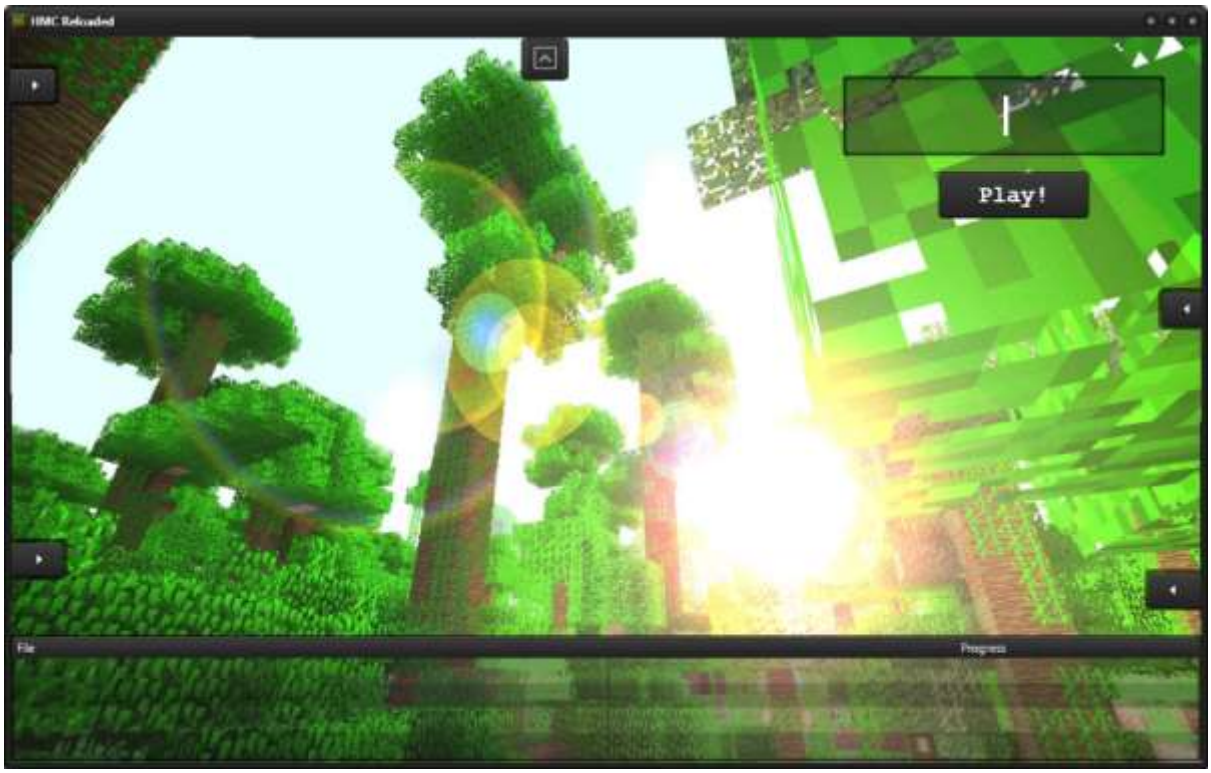
Before

After

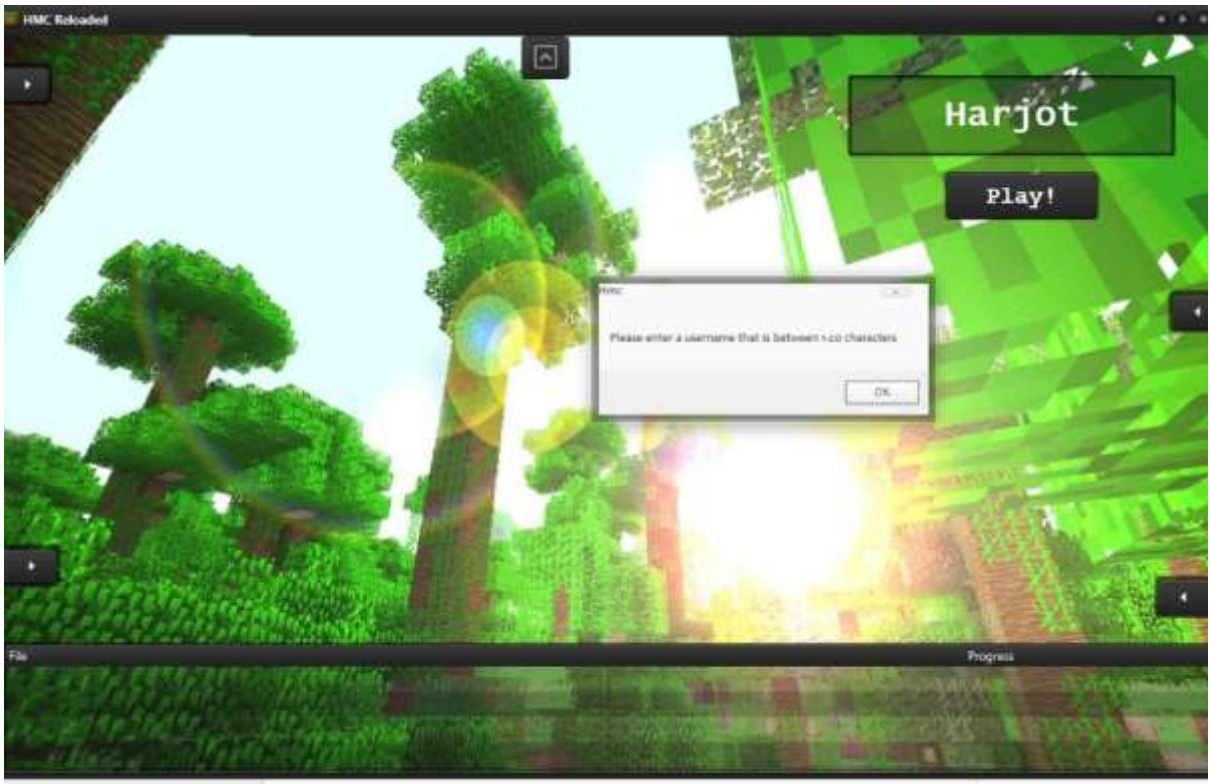


Test 11 - Pass

Before



After



The username is reset, as expected.

Test 12 - Pass

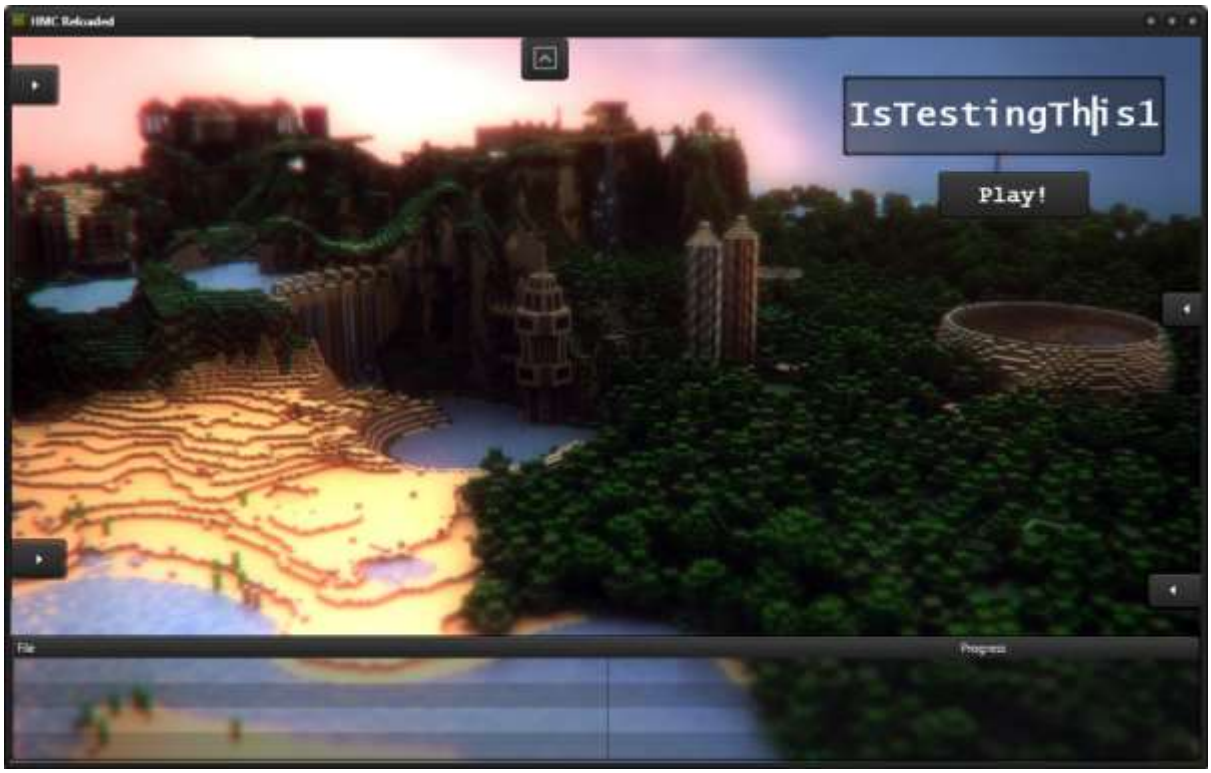
Before

After

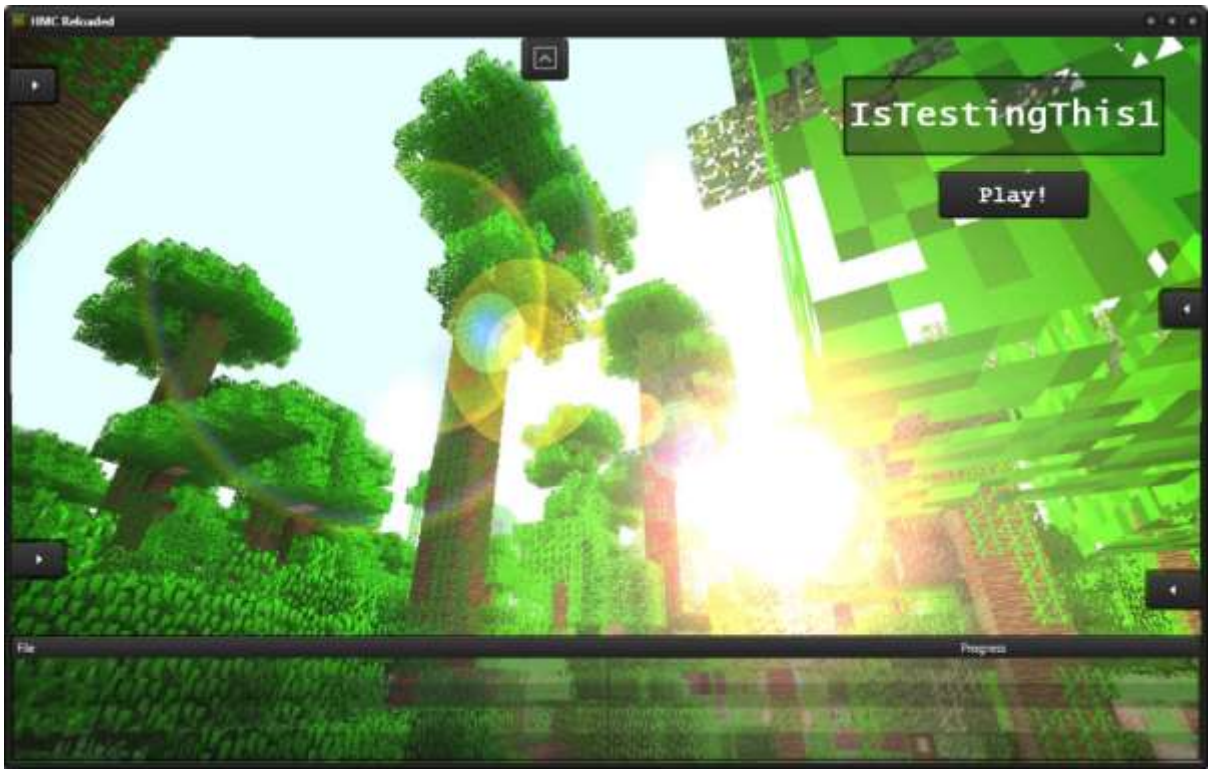
As expected.

Test 13 - Pass

Before



After



As expected, no error message.

Test 14 - Pass

Before

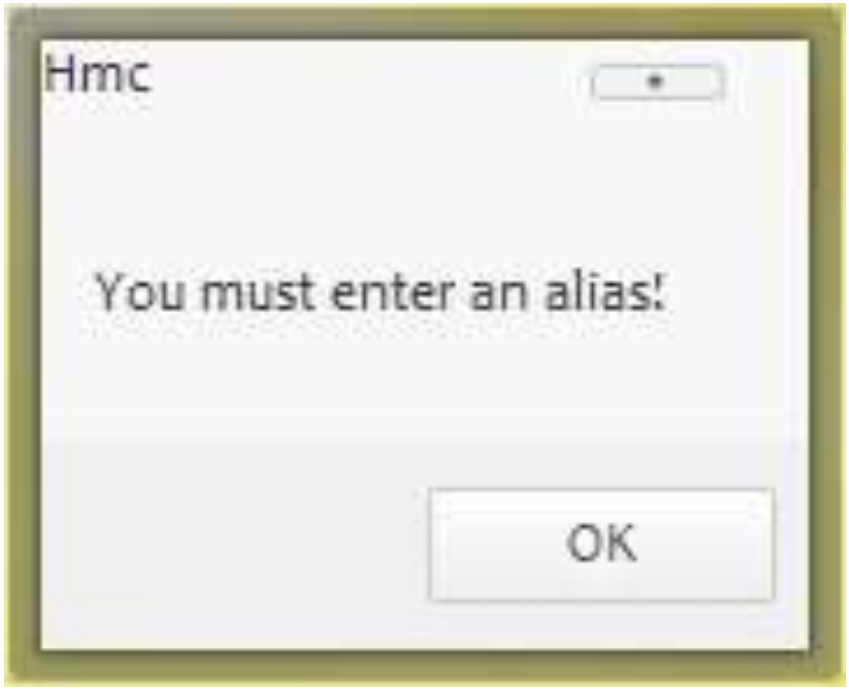
After

The request was sucessful as expected.

Test 15 - Pass

Before

After

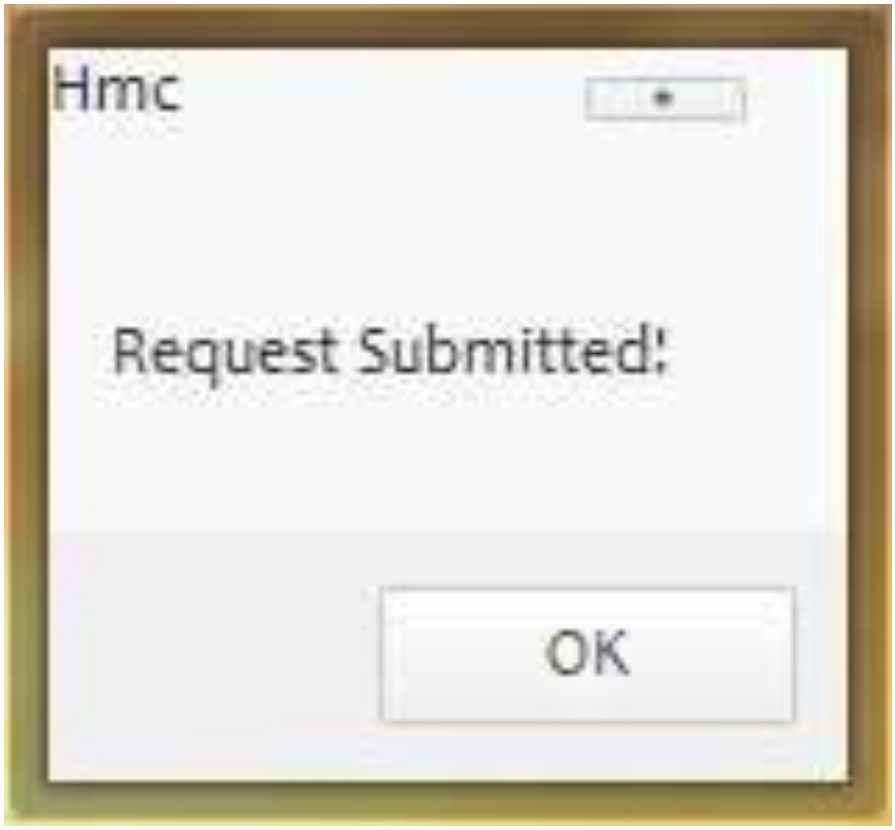


As expected, an error message appears, alerting the user that input must be present.

Test 16 - Pass

Before

After



As expected.

System Maintenance

System Overview

The launcher was programmed using the Delphi programming language, with the Embarcadero XE7 RAD Studio IDE. Firemonkey was the framework/component suite used. Code is organised into units with similar functionality, and abstracted into classes for processes that benefit from universality and non-specific circumstances. The project was written with Windows 32-bit in mind, but successfully executes Minecraft on 64-bit machines that have 64-bit Java installed, and has been written with a view to facilitating modifications that allow compilation for Mac OS X in the future.

There are 11 units in this project:

- UnitMain.pas – All design code and links view/GUI to functionality
- UnitDatabase.pas – Contains all SQL code and database handler class
- UnitDownloader.pas – Multithreaded downloading classes
- UnitLauncher.pas – Non-platform specific classes and methods that help launching Minecraft
- UnitWindows.pas – Windows specific code that uses OS functionality to execute Java
- UnitLogger.pas – Simple universal logger class that the application reports to
- UnitOverlay.pas – Code for frmOverlay, which is the quota timer overlay in-game
- UnitSettings – Settings record, reading/writing to/from executables and settings files
- UnitSync.pas – Classes for comparing JSON files and hashes
- Murmur2.pas – Externally obtained classes modified to calculate a Murmur2 hash from a file
- UnitUpdater.pas – Classes for installing different Minecraft versions in an automated fashion

Non-platform specific code was retained in main units, but any code that had to be platform specific was refactored into a separate unit, so adding functions for Mac OS simply requires that they return the equivalent values the Windows specific functions do.

There are 2 forms. FrmMain contains all of the user interface, and does not contain anything that modifies anything other than the view and appearance of the launcher. This is stored in unitMain.

UnitMain brings all the loosely coupled classes together and links the modules together. This is possible by design – any procedure that a class may need to carry out, but is not part of its functionality, such as downloading when the core functionality is to compare files, is accessible by defining it as a property event. For example, the synchronisation class could have a property called DownloadFile, which is actually a procedure/event that a procedure is assigned to in UnitMain, calling the Download procedure of the TDownloader class. The application is very heavily event-driven, to the extent that custom events are commonly defined as the procedure of object type.

The other form, frmOverlay, is not used as a form, but as an overlay over the Minecraft window. It does this using the Window API to position it in the top centre of the Minecraft window. FrmOverlay could be replaced with native Windows hooks into Minecraft, and directly drawing to the Minecraft window, but is more tedious and error-prone than overlaying.

Each unit has specific functionality inside it. UnitMain is where all the loosely-coupled classes are brought together and tied with callbacks and event procedures.

UnitLauncher is platform independent code that generates the Minecraft command and launches it, with the overlay. Any operating system dependent code is branched into a separate unit. UnitLauncher parses JSON rules and information about how to launch Minecraft, and replace the

arguments and forms a command that can be executed by the operating system, using the Java Runtime Environment's Virtual Machine. JSON may need to be recursively parsed, as information about how to launch Forge depends on information on how to launch vanilla (normal) Minecraft. For liteLoader support, this inherits information from Forge, which inherits information from vanilla Minecraft, thus a recursive parser has been implemented.

Windows specific functions have been isolated into a separate unit, and so writing the same class but for Mac OS X in a different unit with the same names allows for a Mac OS X version to be easily created.

UnitOverlay is a small form that is forced on top of the Minecraft window. Its purpose is to provide a graphic method of designing the overlay.

UnitDatabase contains any database related functionality, and allows unitMain to call queries as if they were actionable procedures. It populates the grid by firing an event that is assigned by UnitMain, in effect a callback function. This function is changed on the outside and so UnitDatabase simply provides data through this medium.

UnitSettings contains the record of settings, and methods for loading and saving from an executable and record of file. This functionality allows the administrator to produce executables with settings attached – no need for additional files, and is more importantly tamper-proof.

UnitDownloads houses a simple queue structure, a threaded downloader, and a download manager which manages a pool of downloaders. Thus the user of this unit can add as many downloads as they wish to the queue, and the manager assigns them to any free downloaders and processes the queue. This pooling system is extremely effective, as it is never known what needs to be downloaded or when. For order, callback functions for beginning and endning a download can be supplied.

UnitSync generates a list of all files inside the Minecraft directory. It then calculates a Murmur2 hash of each file, and saves them in a directory-like JSON. It is able to read two of these JSON structures, and establish what no longer exists on the server, and what needs to be updated on the client i.e. downloads and deletes.

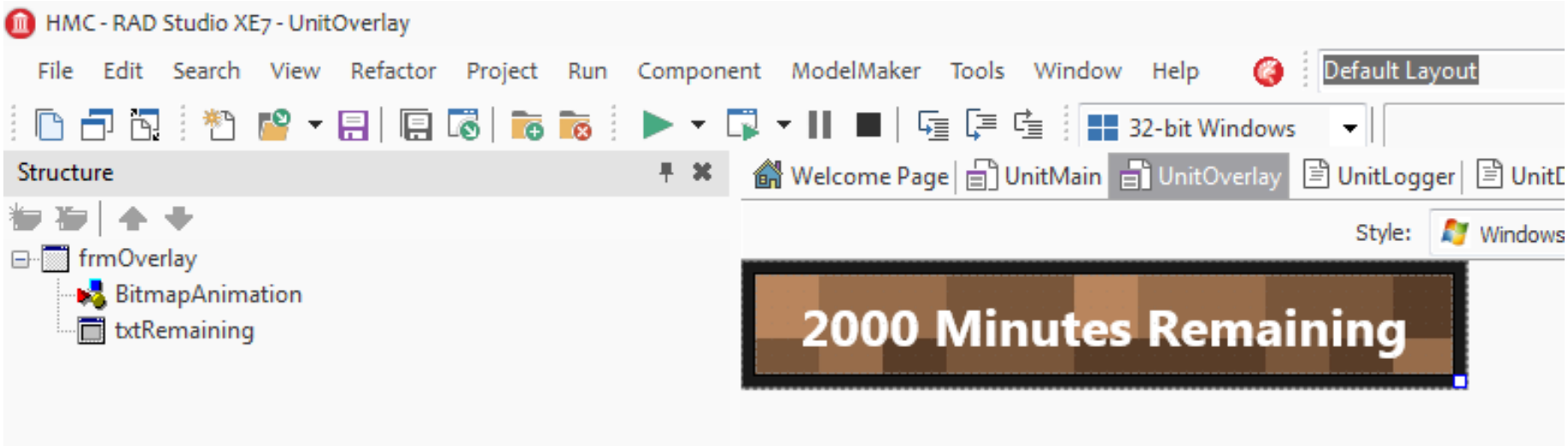
UnitUpdater contains the code that requests a download of the Minecraft version.json from the Mojang servers, and parses it for Minecraft versions. It then is able to download a list of more information about how to download and install any Minecraft version that is 1.6.4 and above, and will queue files to be downloaded. It also creates files that allow Forge or LiteLoader to be installed.

Form Designs

FrmMain

The above screenshot reflects the original prototyping in the design section of the launcher. However, many components are hidden and only rendered visible at runtime. All panes are predesigned, and the arrows are part of the respective panes that they open (which provide similar views to that in the design section by grouping components on a TPanel). The downloads pane has been extended from 2 to 4 slots. The background image is preloaded only for prototyping purposes – images are randomly selected and loaded at runtime. The downloads grid is a TGrid, which provides a progress column and text column, while the database grid is a TStringGrid that only allows for string columns. Although not visible above, the administrative request pane referenced in the design prototypes has been moved into the database pane, as it appeared to be more efficient to use the already existing database pane to display similar information as the user management tab of the database pane does.

FrmOverlay



This form was not part of the design, as it does not appear to the user as something the user interacts with. This form appears directly on top of the Minecraft game, and is borderless, and so is unobtrusive and provides a more native feel. The form itself is composed of a background image, part of the form's properties, a text label that informs the user of their remaining time quota, and finally an animation component that facilities the transformation of the dirt background into an animated lava image.

Further details of form design can be found in the Design section.

Sample of Detailed Algorithm Design

A detailed description of the pseudo-code can be found in the design section of this document.

CompareJSON

The CompareJSON procedure under UnitSync is based on the following pseudo-code:

```
Procedure CompareJSON(Path)
Begin
    Extract ← False
    For keyIndex ← 0 to keys.count – 1 do
    Begin
        KeyName ← MainJSON[Keyindex].Name
        Value ← MainJSON[KeyIndex].Value
        If Exists(OtherJSON[path + keyName] then
            If not (MainJSON[path + keyName] = OtherJSON[path + keyName]) then
                Extract ← True
            else
        else Extract ← True
        if Extract = true then
            if GetDataType(Value) = JSONObject then
                CompareJSON(Path + Key  + '.')
            else
                if GetDataType(Value) = JSONArray then
                    for arrayItem in Value do CompareJSON(Path + Key + '.')
            else
                DifferenceList[DifferenceList.Length] ← Path + Key
        Extract ← False
    end
end
```

The basic purpose of the algorithm is to compare the local file listing under the Minecraft directory to that of the file repository server. This is done by traversing and comparing 2 JSON objects that contain a list of the files on each end, and verifying whether the Murmur2 hashes match, or if the files/keys even exist.

Installing Versions of Minecraft

UnitUpdater is designed on much of the pseudo-code below:

```
LibraryURL ← 'https://libraries.minecraft.net/'
For PackageJSON in JSONFile['libraries'] do
Begin
        KeyName ← PackageJSON['name']
        Package ← CopyUpToColon(KeyName)
        Package ← ReplaceCharacters(Package, '.', '/')
        KeyName ←DeleteUpToColon(KeyName)
        Name ← CopyUpToColon(KeyName)
        Name ← ReplaceCharacters(Name, '.', '/')
        KeyName ← DeleteUpToColon(Keyname)
        Version ← KeyName
DownloadURL ← LibraryURL + Package + '/' +  Name + '/' + Version + '/' + Name + '-' + Version +
'.jar
        NeedsExtracting ← False
        If Exists(PackageJSON['rules']) then
                For RuleJSON in PackageJSON['rules'] do
                Begin
                        If RuleJSON['action'] = 'disallow' then
                                If not(RulesJSON['os.name'] = 'windows')
                                        Download ← True
                        If RuleJSON['action'] = 'allow' then
                                If RulesJSON['os.name'] = 'windows' then
                                        Download ← True
                                Else
                        Else Download ← True
                End
        Else Download ← True
        If Download then DownloadFile
End
```

The algorithm essentially transforms a name of a package under the "name" key in the provided JSON into a downloadable URL, following the conventions laid out by Mojang for downloading these files. See the design for more information.

Launching Minecraft

UnitLauncher is heavily based on the following pseudo-code:

```
MinecraftVersion ←GetCurrentMinecraftVersion
MinecraftDir ←GetMinecraftDirectoryLocation
MinecraftJar ← MinecraftDir + '\versions\' + MinecraftVersion + '\' + MinecraftVersion + '.jar'
MainClass ← JSONFile['mainClass']
Args ← JSONFile['minecraftArguments']
ReplaceArguments(Args)
Command ← GetJavaPath + -Djava.library.path=' + MinecraftDir +'\versions\natives –cp ' +
GetLibraryList(MinecraftVersion) + ';' + MinecraftJar + ' ' + MainClass + ' ' + MinecraftArguments
ExecuteCommand(Command)

Function GetLibraryList (Version : Integer) : String
Begin
        LoadFile(Version + '.json', JSONFile)
        If Exists(JSONFile['inheritsFrom']) then
                GetLibraryList ← GetLibraryList + GetLibraryList(JSONFile['inheritsFrom'])
        GetLibraryList ← GetLibraryList + GetLibraryListFromVersion(Version)
End

Procedure ReplaceArguments(Args : String)
Begin
        Replace('${auth_player_name}, getPlayerUsername)
        Replace('${version_name}', MinecraftVersion)
End
```

The pseudo-code collates all the parameters necessary to launch Minecraft. The design section has more details on this.

Message Logging

Finally, message logging in UnitWindows is heavily influenced by:

```
While Keylogging = true do
Begin
        Message ← GetMessage
        If Length(Message) > 0 then
                SubmitToDatabase(message)
End


Function GetMessage : String
Begin
        Key ← WaitForKeyPress
        If Key = 't' or Key = '/' then
                MessageInProgress ← True
        Repeat
                Key ← WaitForKeyPress
                If Key = RETURN or ESCAPE then
                        MessageInProgress ← False
                Else
                        Message ← Message + Key
                If Key = ESCAPE then
                        Message ← ''
        Until MessageInProgress = false
        GetMessage ← Message
   End


Function WaitForKeypress : Char
Begin
        Repeat
                Pause
        Until getForegroundWindowName = 'Minecraft'
         For char ← A to z do
                If getKeyState(char) = true then
                        WaitForKeyPress ← Char
End
```

Although there are some major changes due to the workings of the Windows API (see code listing). The idea behind the code is to poll each of the keys constantly, and whenever a key press is detected from any of the keys, add it to a message string.

Further comments on these algorithms can be found in the Design section of this project.

Overview of Global Variables

Name	Type	Description
frmMain	TfrmMain	The variable the holds the instance of the form
Log	TLogger	An instance of TLogger, that handles messages and saves logs
DB	TDBHandler	An instance of TDBHandler, which contains all database Functionality
isAdmin	Boolean	If true, used to display admin options
Settings	RConfiguration	A variable that holds the record for storing the settings in
SettingManager	TSettingLoader	An instance of TSettingLoader, which reads and writes records to and from files, and executables
DownloadManager	TDownloadManager	An instance of TDownloadManager, handles all downloading related tasks
JSONGen	TJSONGenerator	An instance of TJSONGenerator, which generates JSON based on the contents of the Minecraft directory
MinecraftUpdater	TMinecraftUpdater	An instance of TMinecraftUpdater, which is a self-contained Minecraft version installer
SyncManager	TSyncManager	An instance of TSyncManager, which decides which files need to be downloaded or deleted
Username	String	Holds in-game Minecraft username
MinecraftDir	String	The location of the Minecraft directory
ZipFile	TZipFile	Instance of TZipFile, used for creating backups of the Minecraft directory, and restoring them
Minecraft	TMinecraftInstance	An instance of TMinecraftInstance, which controls the Minecraft application, and launching

Overview of Procedures and Functions

TfrmMain

Procedure FormCreate(Sender: TObject);

Creates all instances of required objects, and calls Procedures that fetch username. All is executed so that the user does not see anything just yet.

Procedure FormClose(Sender: TObject; var Action: TCloseAction);

Cleans up when the application is about to close. Shuts down any running Minecraft instance, saves the log, and destroys objects.

Procedure btnLogPaneClick(Sender: TObject);

Toggles the Log pane open or closed

Procedure FormDeactivate(Sender: TObject);

When the form loses focus, changes the background image to a random one stored in the resources section of the executable

Procedure FormActivate(Sender: TObject);
Used as a Procedure that is executed when the application has started, AFTER everything has been drawn. Only executed once. Carries out all the checks, initialisation, syncing and initiates all downloads.

Procedure FormResize(Sender: TObject);
Resizes the download grid if the form is resized.

Procedure btnGenerateExecutableClick(Sender: TObject);
Launches a file dialog asking the user where the launcher with embedded settings should be saved to, and generates the launcher.

Procedure cbxDBChange(Sender: TObject);
Enables and disables corresponding settings when the “Enable database support” checkbox is toggled.

Procedure btnApplyDBClick(Sender: TObject);
Connects to the database, verifies and saves database settings.

Procedure btnInitialiseDBClick(Sender: TObject);
Deletes everything in the database, and creates a fresh copy of the table structure.

Procedure ebnDBPasswordClick(Sender: TObject);
Toggles masking the password characters in the database password field.

Procedure btnDefaultJavaClick(Sender: TObject);
Resets all the Java settings to their default values (RAM Settings)

Procedure ttbXmsTracking(Sender: TObject);
Saves the new value of Xms to Settings as the slider is dragged. Ensures that it never exceeds Xmx.

Procedure ttbXmxTracking(Sender: TObject);
Saves the new value of Xmx to settings as the slider is dragged.

Procedure ttbXmxChange(Sender: TObject);
Carries out validation once the user has released control of the slider.

Procedure sbxQuotaTTTChange(Sender: TObject);
Saves the new time till termination after quota value to settings.

Procedure edtFileRepoChange(Sender: TObject);
Saves the file repo URL to settings as long as it validates correctly, and strips the trailing slash.

Procedure edtMCDirChange(Sender: TObject);
Saves the custom Minecraft directory location to settings, as long as it validates correctly, and strips the trailing slash.

Procedure btnRefreshClick(Sender: TObject);
Refreshes the content in the database grids.

Procedure timUsersClick(Sender: TObject);
Loads the content for the Users tab in the database grid when the tab is navigated to.

Procedure btnSaveBackupClick(Sender: TObject);
Saves a backup of the Minecraft directory into a zip file at the location of the user’s choosing.

Procedure btnLoadBackupClick(Sender: TObject);
Restores the backup of the Minecraft directory from the zip file navigated to by the user.

Procedure btnSettingsPaneClick(Sender: TObject);
Toggles the visibility of the settings pane.

Procedure btnPlayClick(Sender: TObject);
Calls Procedures that launch Minecraft, and sets appropriate overlay depending on administrator or user permissions.

Procedure mimAliasClick(Sender: TObject);
Prompts the user for a new alias for the selected user, if the Change Alias item in the popup menu under the User tab in the database pane is clicked

Procedure PopupMenuDBPopup(Sender: TObject);
Focuses the correct row if the user right clicks under the Users tab in the database pane.

Procedure mimBanClick(Sender: TObject);
Bans the selected user if the Ban User item in the popup menu under the User tab in the database pane is clicked.

Procedure mimUnbanClick(Sender: TObject);
Unbans the selected user if the Unban User item in the popup menu under the User tab in the database pane is clicked.

Procedure mimSetAdminClick(Sender: TObject);
Gives administrator permissions the selected user if the Make Admin item in the popup menu under the User tab in the database pane is clicked.

Procedure mimRevokeAdminClick(Sender: TObject);
Revokes administrator permissions the selected user if the Revoke Admin item in the popup menu under the User tab in the database pane is clicked.

Procedure mimGlobalQuotaClick(Sender: TObject);
Prompts the user for a new global time quota, if the Set Global Quota item in the popup menu under the User tab in the database pane is clicked.

Procedure mimQuotaClick(Sender: TObject);
Prompts the user for a new quota for the selected user, if the Change Quota item in the popup menu under the User tab in the database pane is clicked.

Procedure edtSearchChangeTracking(Sender: TObject);
Returns search results in the grid on the current tab for the database pane.

Procedure timSessionsClick(Sender: TObject);
Loads session data from the database if the Sessions tab under the database pane is clicked.

Procedure timMessagesClick(Sender: TObject);
Loads message data from the database if the Messages tab under the database pane is clicked.

Procedure btnDatabasePaneClick(Sender: TObject);
Toggles the opening or closing of the database pane.

Procedure edtNewAliasChange(Sender: TObject);
Validates and submits a new alias request.

Procedure btnSendRequestClick(Sender: TObject);
Submits the alias request to the database, as long as it passes validation.

Procedure btnRequestsUserPaneClick(Sender: TObject);
Toggles the Requests pane open or closed.

Procedure timRequestsClick(Sender: TObject);
Loads Request data from the database if the Request tab under the database pane is clicked.

Procedure PopupMenuRequestsPopup(Sender: TObject);
Displays menu items depending on whether a request or mod was right clicked in the Requests tab under the database pane.

Procedure mimAcceptAliasClick(Sender: TObject);
If the Accept Alias item on the popup menu under the Requests tab in the database is pressed, automatically change the selected user's name to that requested.

Procedure mimDeclineAliasClick(Sender: TObject);
If the Decline Alias item on the popup menu under the Requests tab in the database is pressed, ignore and hide the request permanently.

Procedure MimProcessRequestClick(Sender: TObject);
If the Hide item on the popup menu under the Requests tab in the database is pressed, hide the item permanently.

Procedure cbxCopyToChange(Sender: TObject);
If the checkbox Automatically Copy is checked, a file dialog will appear requesting the automatic copy location of the Minecraft directory when the Sync Files button is pressed.

Procedure grdDownloadsGetValue(Sender: TObject; const Col, Row: Integer; var Value: TValue);
Populates the download grid with URL and progress columns.

Procedure btnInstallClick(Sender: TObject);
Installs the version of Minecraft specified in the drop down box, to the Minecraft directory.

Procedure btnUpdatesPaneClick(Sender: TObject);
Toggles the visibility of the Updates pane.

Procedure grdLogDrawColumnCell(Sender: TObject; const Canvas: TCanvas; const Column: TColumn; const Bounds: TRectF; const Row: Integer; const Value: TValue; const State: TGridDrawStates);
Draws the red and orange background for error and warning messages.

Procedure btnSyncModsClick(Sender: TObject);
Initiates the syncing of mods process.

Procedure mmoRegexChange(Sender: TObject);
Ensures that there are no trailing blank lines in the Exclusion Regex memo.

Procedure btnSyncPaneClick(Sender: TObject);
Toggles the visibility of the Sync pane.

Procedure OnLog(Text : String; EntryType : TEntryType);
Logs text to the grid when the TLogger object fires this event.

Procedure OpenLogPane;
Animates the log pane open.

Procedure CloseLogPane;
Animates the log pane closed.

Procedure OpenDatabasePane;
Animates the database pane open.

Procedure CloseDatabasePane;
Animates the database pane closed.

Procedure OpenSettingsPane;
Animates the settings pane open.

Procedure CloseSettingsPane;
Animates the settings pane closed.

Procedure OpenUpdatesPane;
Animates the updates pane open.

Procedure CloseUpdatesPane;
Animates the Updates pane closed

Procedure OpenSyncPane;
Animates the sync pane open.

Procedure CloseSyncPane;
Animates the sync pane closed.

Procedure OpenRequestsUserPane;
Animates the requests pane open.

Procedure CloseRequestsUserPane;
Animates the request pane closed.

Procedure RandomiseBackground;
Loads a random background embedded in the executable file.

Procedure OnBan(Sender : TObject);
Fired when the database handler detects the user has been banned, terminates the user's Minecraft session, and displays animated banned text on screen.

Procedure edtNameChange(Sender: TObject);
Validates and sets the new alias for an administrator that manually changes the textbox.

Procedure PopulateUsers(Sender : TObject; Fields : TFields);
Populates the Users grid with data from the database handler.

Procedure PopulateSessions(Sender : TObject; Fields : TFields);
Populates the Sessions grid with data from the database handler.

Procedure PopulateMessages(Sender : TObject; Fields : TFields);
Populates the Messages grid with data from the database handler.

Procedure PopulateRequests(Sender : TObject; Fields : TFields);
Populates the Requests grid with data from the database handler.

Procedure OnMessageSend(Sender : TObject; Text : String);
Fired when a message is sent from Minecraft, logs this, and passes it to the database handler to insert into the database.

Procedure OnMinecraftClose(Sender : TObject);
Fired when Minecraft closes.

Procedure OnQuotaChange(Sender : TObject; MinutesLeft : Integer);
Fired when the database handler detects the quota for the user has changed.

Procedure OnQuotaUp(Sender : TObject);
Fired when the database handler reports that the current user has no more Minecraft time left.

Procedure OnDownloadProgress(Sender : TObject; URL : String; Progress : Integer);
Populates download grid with progress and URL, fired by downloader with relevant information.

Procedure OnDownloadFinish(Sender : TObject; URL : String);
Clears current URL from grid when download is finished, fired by downloader.

Procedure OnDownloadBegin(Sender : TObject; URL : String);
Fired by downloader when the download of URL is about to begin.

Procedure OnNewDownload(Sender : TObject; URL : String; FilePath : String; OnComplete : TDownloadEvent);
Links downloading from other classes, which call this Procedure indirectly.

Procedure AddDownload(URL : String; FilePath : String; OnComplete : TDownloadEvent = nil);
Adds a download to the download queue.

Procedure OnGetVersionsList(Sender : TObject; List : TStringList);
Fired when the Minecraft updater has returned and parsed a list of Minecraft versions that are 1.6.4 or above.

Procedure OnQueueEmpty(Sender : TObject);
Fired when the download queue is empty, and enables play button.

Procedure OnUpdateLauncher(Sender : TObject);
Fired when the sync manager detects there is a launcher update available, and so evaluates whether it is possible to update the launcher properly.

Procedure OnUpdateDownloaded(Sender : TObject; URL : String);
Fired when the launcher update has downloaded, so the relevant message is displayed.

Procedure SetHints;
Sets all the hints for each of the pane toggle buttons.

Procedure CreateMySQLLib;
Extracts the libMySQL.dll library file into the launcher directory from the executable if it is not detected.

TLogger

Procedure Error(Text : String);

Called by other parts of the program, and fires the OnLog event with the Error type.

Procedure Info(Text : String);

Called by other parts of the program, and fires the OnLog event with the Warning type.

Procedure Warn(Text : String);

Called by other parts of the program, and fires the OnLog event with the Info type.

TDBHandler

Constructor Create(AOwner: TComponent);

Creates the database and timers used by the class.

Destructor Destroy;

Destroys all the created timers and objects.

Procedure InitialiseDB;

Deletes the HMC database, creates a new HMC database, and creates all the relevant tables.

Function DBExists: Boolean;

Returns true if HMC exists.

Function DBServerExists: Boolean;

Connects to the database and returns true if this was successful.

Procedure SetDatabase(DBURL : String; Username: String; Password: String);

Sets the database URL, username, and password.

Function GetUserID(LoginName : String) : Boolean;

Gets the user's id based on their Windows login name, and store it in LocalUserID. Used for every query that doesn't take a user id.

Function GetAlias : String;

Get the user's alias given that getUserID has been called before.

Function isAdmin : Boolean;

Returns true if LocalUserID is an administrator.

Function isBanned: Boolean;

Returns true if LocaluserID is banned.

Procedure BeginSession(ComputerName : String = '');

Begins a new session with localUserID at the current time, and optional computer name.

Procedure EndSession();

Ends the session that was created with BeginSession.

Procedure InsertMessage(MessageText : String);

Inserts a message with the MessageText into the database at the current time, with LocalUserID.

Procedure SetAlias(UserID : Integer; Alias : String);

Sets the Alias of User ID to Alias.

Procedure RequestAlias(Alias : String);
Puts a request in the database with LocalUserID wanting Alias.

Procedure SetAdmin(UserID : Integer);
Sets Admin to true for UserID.

Procedure RevokeAdmin(UserID : Integer);
Set Admin to false for UserID

Procedure SetBanned(UserID : Integer);
Set Banned to true for UserID.

Procedure ProcessRequest(RequestID : Integer);
Set Hide to true for RequestID.

Procedure SetUnbanned(UserID : Integer);
Set Banned to false for UserID.

Procedure RequestMod(ModRequest : String);
Request mod from LocalUserID at current timestamp, with ModRequest as the value of the request.

Function GetAverageTime(UserID : Integer) : Single;
Returns average time that User ID spends playing Minecraft.

Procedure GetMessages;
Queries for all messages, and returns them using ProcessQuery.

Procedure GetUsers;
Queries for all users, and returns them using ProcessQuery.

Procedure GetSessions;
Queries for all Session data, and returns them using ProcessQuery.

Procedure GetRequests;
Queries all Request data, and returns them using ProcessQuery.

Procedure CreateUser(LoginName : String);
Create a new user with LoginName.

Procedure EnableBanChecker;
Begin checking whether LocalUserID is banned or not.

Procedure EnableQuota;
Enable decrementing the quota for LocalUserID

Procedure ChangeDefaultQuota(Minutes : Integer);
Changes the default quota to Minutes, used by users when they log on the next day.

Function RequestType(RequestTypeID : Integer) : String;
Returns the request type given the RequestTypeID.

Function GetQuota : Integer;
Returns the quota for LocalUserID.

Procedure SetQuota(UserID : Integer; Quota : Integer);
Sets the quota to Quota minutes for UserID.

Function GetDefaultQuota : Integer;
Gets the default user quota.

Procedure SearchUser(Query : String);
Searches different fields of the User table with Query.

Procedure SearchSessions(Query : String);
Searches different fields of the Sessions table with Query.

Procedure SearchMessages(Query : String);
Searches different fields of the Messages table with Query.

Procedure SearchRequests(Query : String);
Searches different fields of the Requests table with Query.

Procedure OnDBError(Sender: TObject; const Initiator : IFDStanObject; var Error : Exception);
Fired if there is an error with the database and handles it gracefully.

Procedure OnBannedCheckTimer(Sender : TObject);
Checks every second whether LocalUserID is banned or not.

Procedure OnQuotaTimer(Sender : TObject);
Decrements and updates Quota for LocalUserID every minute.

Procedure ProcessQuery;
Procedure used by all queries that return a dataset to facilitate using the same callback.

TQueue

constructor Create;
Creates the Queue structure.

Procedure Enqueue(QueueItem : RQueueItem);
Adds an item to the array of RQueueItem.

Function Dequeue : RQueueItem;
Returns the first element of the array and shifts the queue downwards.

Function High : Integer;
Returns the high index of the array.

Function Low : Integer;
Returns the low index of the array.

Function Length : Integer;
Returns the length of the array.

TDownloader

Procedure OnHTTPWork(Sender : TObject; WorkMode : TWorkMode; WorkCount : Int64);
Fired whenever a downloader downloads a chunk. Used to update progress bars.

Procedure OnFinish(Sender : TObject; WorkMode : TWorkMode);
Fired when the downloader has finished. Used to fire an event that propagates to frmMain and cleans up grids.

Procedure Execute; Override;

The procedure that is executed by a newly created thread i.e. any code here is executed in a multithreaded fashion once.

Procedure DownloadFile;

Downloads the file specified in CurrentURL to CurrentPath.

Constructor Create;

Creates the threaded downloader.

Procedure Download(URL : String; Path : String);

Sets the CurrentUrl and CurrentPath private variables to URL and Path, and downloads it in a multithreaded fashion.

TDownloadManager

constructor Create;

Sets a length on the number of simultaneous downloads.

destructor Destroy;

Cleans up downloaders.

Procedure AddDownload(URL : String; FilePath : String; OnBegin : TDownloadEvent = nil; OnProgressChange : TDownloadProgressEvent = nil; OnDownloadComplete : TDownloadEvent = nil; OnDownloadError : TDownloadErrorEvent = nil; OnDownloadCompleteSecondary : TDownloadEvent = nil);

Adds a download and fires the relevant events in call back style.

Function DownloadCount : Integer;

Returns number of active downloads.

Procedure ProcessQueue;

Used to check whether there are any items on the queue, and whether there are any downloaders free to carry out a download.

Procedure OnDownloadErrorRetry(Sender : TObject; URL : String; Path : String; E : Exception; OnBegin : TDownloadEvent; OnProgressChange : TDownloadProgressEvent; OnDownloadComplete : TDownloadEvent; OnDownloadCompleteSecondary : TDownloadEvent = nil);

Fired when a download failed once. Retries the download.

Procedure OnDownloadErrorFail(Sender : TObject; URL : String; Path : String; E : Exception; OnBegin : TDownloadEvent; OnProgressChange : TDownloadProgressEvent; OnDownloadComplete : TDownloadEvent; OnDownloadCompleteSecondary : TDownloadEvent = nil);

Fired after second download failure. Does not retry download and logs an error.

Procedure OnNewDownload(Sender : TObject);

Fired by downloader when it is ready to download another file. Calls ProcessQueue.

TProgram

Procedure CheckTermination(Sender : TObject);

Checks whether Minecraft has terminated. Fires event if true.

Procedure CheckKeyPress(Sender : TObject);

Fired every 5-10 milliseconds to poll any keys that have been pressed.

Procedure SetOverlayPosition(Sender : TObject);
Positions overlay to be in top centre of Minecraft window.

Function Launch(Command : String) : Boolean;
Executes the given command using CreateProcess (Windows API)

Procedure Close;
Closes the program, first gracefully, otherwise terminates it.

Constructor Create;
Creates the timers and self.

Destructor Destroy;
Cleans up the instance.

Procedure EnableRectangle;
Enables the overlay.

TCommandGenerator

Function GenerateCommand : String;
Returns the generated command required to launch Minecraft.

Procedure ReplaceArguments;
Replaces all the arguments in the Minecraft command line with their respective values by calling each Replace-type function.

Procedure ReplaceArch;
Replaces argument with architecture of OS.

Procedure ReplaceUsername;
Replaces username argument with provided username.

Procedure ReplaceVersionName;
Replaces version name argument with version name of Minecraft.

Procedure ReplaceGameDirectory;
Replace game directory argument with location of Minecraft directory.

Procedure ReplaceAssetsRoot;
Replace assets root argument with location of the assets root inside the Minecraft directory.

Procedure ReplaceGameAssets;
Replaces the gameAssets folder with the legacy folder inside the Minecraft directory.

Procedure ReplaceAssetIndexName;
Replace the assetInedxname with that provided in the information JSON provided by the Minecraft server.

Procedure ReplaceAuthUUID;
Replace AuthUUID with an invalid random string. Should ideally replace it with authetntication code retrieved from Mojang servers after signing in with a Minecraft account.

Procedure ReplaceAccessToken;
Replace AccessToke with acquired token provided by Mojang during login process. Actually places invalid token instead.

Procedure ReplaceUserProperties;
Replace UserProperties with an empty object {}.

Procedure ReplaceUserType;
Replace UserType with any string.

Function Libraries(JSON : ISuperObject) : String;
Returns all the paths of the required libraries required to launch Minecraft, all separated with semi-colons. Recursive function that calls itself if the inheritsFrom property is found, as the same must be done for JSON that inherits from other JSON.

Function MainClass : String;
Returns the name of the Main java class to be executed.

Function MinecraftArguments : String;
Returns a sanitised and parsed version of the Minecraft arguments.

Function MinecraftJar : String;
Returns the location of the Minecraft jar to be executed.

TMinecraftInstance

Constructor Create;
Creates an instance.

Destructor Destroy;
Cleans up the instance.

Procedure Launch(Username : String; Xms : Integer; Xmx : Integer; MinecraftDir : String);
Launches Minecraft given a username, starting RAM, maximum RAM, and the Minecraft directory.

Procedure SetMessageBox(Text : String);
Sets the text on the overlay window.

Procedure SetLavaOverlay;
Changes the background of the quota overlay to an animated lava image.

Procedure TerminateIn(Seconds : Integer);
Begins a countdown on the overlay, after which Minecraft is terminated.

Procedure Close;
Closes Minecraft.

Procedure ChainOnClose(Sender : TObject);
Connects the OnClose event from TProgram to TfrmMain.

Procedure RecordMessage(Sender : TObject; Key : Char);
Given a keystroke, record a message based on whether t or ‘/’ is pressed.

Procedure CountDown(Sender : TObject);
Begin the countdown on the overlay.

TfrmOverlay

Procedure ChangeToLava;
Loads the lava overlay from form.

Procedure SetText(Text : String);
A setter function that sets the text for the label on the form.

TMinecraftUpdater

Constructor Create;
Creates an instance.

Destructor Destroy;
Cleans up instance.

Procedure GetVersions;
Gets a list of all Minecraft version 1.6.4 and above, stable release only.

Procedure Install(Version : String);
Installs a Minecraft version given Version.

Procedure OnDownloadVersionsJSON(Sender : TObject; URL : String);
Fired when the download of Versions.json containing the Minecraft versions is downloaded. Parses the file.

Procedure OnDownloadMinecraftJar(Sender : TObject; URL : String);
Fired when the main Minecraft Jar is downloaded.

Procedure OnDownloadVersionInformation(Sender : TObject; URL : String);
Fired when information about what to download for a specific Minecraft verison is downloaded, and the correct files for the current platform are downloaded.

Procedure OnDownloadAssetInformation(Sender : TObject; URL : String);
Fired when asset location information is downloaded, and queues up all the assets for download.

Procedure OnCompressedFile(Sender : TObject; URL : String);
Fired when a file needs extracting.

TJSONGenerator = Class

Procedure SaveTo(FileName : String);
Saves the generated JSON to Filename.

Procedure Generate(MinecraftDir : String; ExclusionList : TStrings);
Generates a list of all files in the Minecraft directory, along with a murmur2 hash corresponding to the file. Excludes any file that matches any of the Regexes held in ExlcusionList.

Procedure AddFile(FileName : String);
Adds a file to the generated JSON.

TFileSyncManager = Class

Constructor Create;
Creates an instance.

Destructor Destroy;
Cleans up instance.

Procedure Compare;
Compares LocalList against RemoteList to find out what to download, and then vice-versa to find out what to delete.

Procedure OnServerFilesDownload(Sender : TObject; URL : String);
Fired when Files.json has been downloaded from the File Repo server, and so compare() can be called.

Procedure CompareJSON(MainJSON : ISuperObject; SubJSON : ISuperObject; const SObject: TSuperTableString = nil; const OnDifference : TJSONDifferenceEvent = nil; const path: string = '');
Recursive algorithm that traverses the composite directory-structure JSON, and compares it with the provided JSON.

Procedure OnDownloadFile(Sender : TObject; Path : String);
Fired when a file needs to be downloaded by CompareJSON.

Procedure OnDeleteFile(Sender : TObject; Path : String);
Fired when a file needs to be deleted by CompareJSON.

Code Listing

UnitMain

```
unit UnitMain;


interface


uses UnitLogger, UnitDatabase, UnitSettings, UnitWindows, UnitLauncher,
UnitDownloader, UnitUpdater, UnitSync,

  System.Rtti, FireDAC.Phys.MySQLDef,

  FireDAC.UI.Intf, FireDAC.FMXUI.Wait, FireDAC.FMXUI.Async, FMX.Ani,
FMX.graphics,

  FMX.Effects, FMX.Objects, FMX.StdCtrls, FireDAC.Comp.UI,

   FireDAC.Phys.MySQL, FMX.Controls, FMX.Grid, FMX.TabControl,

  FMX.Layouts, FMX.Edit, System.Classes, FMX.Types,

  System.SysUtils, System.Types, System.UITypes, System.Variants,
Data.DB, FMX.Forms, FMX.dialogs, System.Math, Windows,

  Data.Bind.EngExt, Fmx.Bind.DBEngExt, System.Bindings.Outputs,

  Fmx.Bind.Editors, Data.Bind.Components, FMX.EditBox, FMX.SpinBox,
FMX.Menus, System.Zip,

  FMX.ListBox, FMX.Memo, FireDAC.Stan.Intf, FireDAC.Phys,

  FMX.Controls.Presentation, HS_FMXHints,IOUtils;


type

  TfrmMain = class(TForm)

    imgBackground: TImage;

    StyleBook: TStyleBook;

    btnLogPane: TButton;

    edtName: TEdit;

    btnPlay: TButton;

    grdLog: TStringGrid;

    pnlLog: TPanel;

    pnlSettings: TPanel;

    tclSettings: TTabControl;

    timDatabase: TTabItem;

    timJava: TTabItem;

    timStudent: TTabItem;
```

```
grdDownloads: TGrid;
URLColumn: TStringColumn;
ProgressColumn: TProgressColumn;
StringColumn: TStringColumn;
FDPhysMySQLDriverLink: TFDPhysMySQLDriverLink;
FDGUIxWaitCursor: TFDGUIxWaitCursor;
FDGUIxAsyncExecuteDialog: TFDGUIxAsyncExecuteDialog;
lblQuota: TLabel;
timBackup: TTabItem;
pnlSync: TPanel;
pnlDB: TPanel;
pnlRequestUser: TPanel;
txtBanned: TText;
BlurEffectBanned: TBlurEffect;
FloatAnimationBanned: TFloatAnimation;
AniBusy: TAniIndicator;
btnGenerateExecutable: TButton;
SaveExeDialog: TSaveDialog;
edtDBURL: TEdit;
cbxDB: TCheckBox;
lblDBUsername: TLabel;
lblDBURL: TLabel;
lblDBPassword: TLabel;
edtDBPassword: TEdit;
edtDBUsername: TEdit;
btnInitialiseDB: TButton;
btnApplyDB: TButton;
btnDefaultJava: TButton;
ebnDBPassword: TEditButton;
lblXmsInfo: TLabel;
ttbXms: TTrackBar;
lblXms: TLabel;
lblXmsMB: TLabel;
lblXmxInfo: TLabel;
```

```
lblXmx: TLabel;
lblXmxMB: TLabel;
ttbXmx: TTrackBar;
lblQuotaTTT: TLabel;
sbxQuotaTTT: TSpinBox;
lblCustomMCDirInfo: TLabel;
lblModRepoInfo: TLabel;
edtFileRepo: TEdit;
edtMCDir: TEdit;
btnSaveBackup: TButton;
btnLoadBackup: TButton;
SaveBackupDialog: TSaveDialog;
OpenBackupDialog: TOpenDialog;
tclDB: TTabControl;
timUsers: TTabItem;
timSessions: TTabItem;
timMessages: TTabItem;
btnRefresh: TButton;
sclLoginName: TStringColumn;
sclAlias: TStringColumn;
sclBanned: TStringColumn;
sclAdmin: TStringColumn;
sgdUsers: TStringGrid;
sclAverageTime: TStringColumn;
sclNumberOfSessions: TStringColumn;
sclQuotaRemaining: TStringColumn;
PopupMenuDB: TPopupMenu;
mimBan: TMenuItem;
mimUnban: TMenuItem;
mimQuota: TMenuItem;
mimGlobalQuota: TMenuItem;
mimSetAdmin: TMenuItem;
mimRevokeAdmin: TMenuItem;
mimAlias: TMenuItem;
```

```
pnlUpdates: TPanel;
btnSettingsPane: TButton;
sclHiddenID: TStringColumn;
edtSearch: TEdit;
SgdSessions: TStringGrid;
SclSessionsLoginName: TStringColumn;
SclSessionStart: TStringColumn;
SclSessionEnd: TStringColumn;
sclComputerName: TStringColumn;
SgdMessages: TStringGrid;
SclMessagesLoginName: TStringColumn;
SclTimeSent: TStringColumn;
SclMessage: TStringColumn;
btnDatabasePane: TButton;
tclUserRequest: TTabControl;
btnRequestsUserPane: TButton;
btnSendRequest: TButton;
timRequestAlias: TTabItem;
timRequestMod: TTabItem;
edtNewAlias: TEdit;
lblReborn: TLabel;
edtModRequest: TEdit;
lblIWant: TLabel;
lblBecause: TLabel;
mmoBecause: TMemo;
timRequests: TTabItem;
SgdRequests: TStringGrid;
SclLoginNameRequests: TStringColumn;
SclDateRequest: TStringColumn;
SclRequestValue: TStringColumn;
SclRequestType: TStringColumn;
SclRequestID: TStringColumn;
SclUserID: TStringColumn;
PopupMenuRequests: TPopupMenu;
```

```
mimAcceptAlias: TMenuItem;
mimDeclineAlias: TMenuItem;
MimProcessRequest: TMenuItem;
btnSyncMods: TButton;
lblRegexExclude: TLabel;
mmoRegex: TMemo;
cbxCopyTo: TCheckBox;
OpenDialogSync: TOpenDialog;
cmbVersions: TComboBox;
lblVersion: TLabel;
btnInstall: TButton;
btnUpdatesPane: TButton;
btnSyncPane: TButton;
BindingsList: TBindingsList;
LinkControlToPropertyEnabled: TLinkControlToProperty;
LinkControlToPropertyEnabled2: TLinkControlToProperty;
LinkControlToPropertyEnabled3: TLinkControlToProperty;
LinkControlToPropertyEnabled4: TLinkControlToProperty;
LinkControlToPropertyEnabled5: TLinkControlToProperty;
LinkControlToPropertyEnabled6: TLinkControlToProperty;
LinkControlToPropertyEnabled7: TLinkControlToProperty;
LinkControlToPropertyEnabled8: TLinkControlToProperty;
Procedure FormCreate(Sender: TObject);
Procedure FormClose(Sender: TObject; var Action: TCloseAction);
Procedure btnLogPaneClick(Sender: TObject);
Procedure FormDeactivate(Sender: TObject);
Procedure FormActivate(Sender: TObject);
Procedure FormResize(Sender: TObject);
Procedure btnGenerateExecutableClick(Sender: TObject);
Procedure cbxDBChange(Sender: TObject);
Procedure btnApplyDBClick(Sender: TObject);
Procedure btnInitialiseDBClick(Sender: TObject);
Procedure ebnDBPasswordClick(Sender: TObject);
Procedure btnDefaultJavaClick(Sender: TObject);
```

```
Procedure ttbXmsTracking(Sender: TObject);
Procedure ttbXmxTracking(Sender: TObject);
Procedure ttbXmxChange(Sender: TObject);
Procedure sbxQuotaTTTChange(Sender: TObject);
Procedure edtFileRepoChange(Sender: TObject);
Procedure edtMCDirChange(Sender: TObject);
Procedure btnRefreshClick(Sender: TObject);
Procedure timUsersClick(Sender: TObject);
Procedure btnSaveBackupClick(Sender: TObject);
Procedure btnLoadBackupClick(Sender: TObject);
Procedure btnSettingsPaneClick(Sender: TObject);
Procedure btnPlayClick(Sender: TObject);
Procedure mimAliasClick(Sender: TObject);
Procedure PopupMenuDBPopup(Sender: TObject);
Procedure mimBanClick(Sender: TObject);
Procedure mimUnbanClick(Sender: TObject);
Procedure mimSetAdminClick(Sender: TObject);
Procedure mimRevokeAdminClick(Sender: TObject);
Procedure mimGlobalQuotaClick(Sender: TObject);
Procedure mimQuotaClick(Sender: TObject);
Procedure edtSearchChangeTracking(Sender: TObject);
Procedure timSessionsClick(Sender: TObject);
Procedure timMessagesClick(Sender: TObject);
Procedure btnDatabasePaneClick(Sender: TObject);
Procedure edtNewAliasChange(Sender: TObject);
Procedure btnSendRequestClick(Sender: TObject);
Procedure btnRequestsUserPaneClick(Sender: TObject);
Procedure timRequestsClick(Sender: TObject);
Procedure PopupMenuRequestsPopup(Sender: TObject);
Procedure mimAcceptAliasClick(Sender: TObject);
Procedure mimDeclineAliasClick(Sender: TObject);
Procedure MimProcessRequestClick(Sender: TObject);
Procedure cbxCopyToChange(Sender: TObject);
```

```
    Procedure grdDownloadsGetValue(Sender: TObject; const Col, Row:
Integer;
      var Value: TValue);
    Procedure btnInstallClick(Sender: TObject);
    Procedure btnUpdatesPaneClick(Sender: TObject);
    Procedure grdLogDrawColumnCell(Sender: TObject; const Canvas:
TCanvas;
      const Column: TColumn; const Bounds: TRectF; const Row: Integer;
      const Value: TValue; const State: TGridDrawStates);
    Procedure btnSyncModsClick(Sender: TObject);
    Procedure mmoRegexChange(Sender: TObject);
    Procedure btnSyncPaneClick(Sender: TObject);
  private
    //Defined for the download grid for 2 columns
    DProgress : Array [0..DOWNLOADTHREADS - 1] of Integer;
    Downloads : Array [0..DOWNLOADTHREADS - 1] of String;
  public
    Procedure OnLog(Text : String; EntryType : TEntryType);
    Procedure OpenLogPane;
    Procedure CloseLogPane;
    Procedure OpenDatabasePane;
    Procedure CloseDatabasePane;
    Procedure OpenSettingsPane;
    Procedure CloseSettingsPane;
    Procedure OpenUpdatesPane;
    Procedure CloseUpdatesPane;
    Procedure OpenSyncPane;
    Procedure CloseSyncPane;
    Procedure OpenRequestsUserPane;
    Procedure CloseRequestsUserPane;
    Procedure RandomiseBackground;
    Procedure OnBan(Sender : TObject);
    Procedure edtNameChange(Sender: TObject);
    Procedure PopulateUsers(Sender : TObject; Fields : TFields);
    Procedure PopulateSessions(Sender : TObject; Fields : TFields);
```

```
    Procedure PopulateMessages(Sender : TObject; Fields : TFields);
    Procedure PopulateRequests(Sender : TObject; Fields : TFields);
    Procedure OnMessageSend(Sender : TObject; Text : String);
    Procedure OnMinecraftClose(Sender : TObject);
    Procedure OnQuotaChange(Sender : TObject; MinutesLeft : Integer);
    Procedure OnQuotaUp(Sender : TObject);
    Procedure OnDownloadProgress(Sender : TObject; URL : String; Progress
: Integer);
    Procedure OnDownloadFinish(Sender : TObject; URL : String);
    Procedure OnDownloadBegin(Sender : TObject; URL : String);
    Procedure OnNewDownload(Sender : TObject; URL : String; FilePath :
String; OnComplete : TDownloadEvent);
    Procedure AddDownload(URL : String; FilePath : String; OnComplete :
TDownloadEvent = nil);
    Procedure OnGetVersionsList(Sender : TObject; List : TStringList);
    Procedure OnQueueEmpty(Sender : TObject);
    Procedure OnUpdateLauncher(Sender : TObject);
    Procedure OnUpdateDownloaded(Sender : TObject; URL : String);
    Procedure SetHints;
    Procedure CreateMySQLLib;
  end;
var
  frmMain: TfrmMain;
  Log : TLogger;
  DB : TDBHandler;
  isAdmin : Boolean;
  Settings : RConfiguration;
  SettingManager : TSettingLoader;
  DownloadManager : TDownloadManager;
  JSONGen : TJSONGenerator;
  MinecraftUpdater : TMinecraftUpdater;
  SyncManager : TFileSyncManager;
  Username : String;
  MinecraftDir : String;
  ZipFile : TZipFile;
```

```
  Minecraft : TMinecraftInstance;

Const
  //Change if new images have been added to resources
  NumberOfBackgroundImages = 13;
  //Change this if the code changes and for pushing out an update
  LauncherVersion = 15;

implementation

{$R *.fmx}

{$REGION 'Layout Related'}
//Handle resizing to correct proportions
Procedure TfrmMain.FormResize(Sender: TObject);
begin
  grdDownloads.Columns[0].Width := (Width -16) / 2;
  grdDownloads.Columns[1].Width := (Width -16) / 2;
  grdDownloads.UpdateColumns;
  pnlDB.Position.Y := 1-pnlDB.Height;
end;

//Randomize background image
Procedure TfrmMain.RandomiseBackground;
  var
  LayoutID : Integer;
  RStream : TResourceStream;
begin
//Choose a random image out of those available and load it
  Randomize;
  LayoutID := RandomRange(1, NumberOfBackgroundImages + 1);
  RStream := TResourceStream.Create(MainInstance, 'BG' +
IntToStr(LayoutID), RT_RCDATA);
  imgBackground.MultiResBitmap.LoadItemFromStream(RStream, 1.000);
```

```
  RStream.Free;
  Log.Info('Layout ' + IntToStr(LayoutID) + ' chosen');
end;

//Toggle Log Pane Open
Procedure TfrmMain.btnLogPaneClick(Sender: TObject);
begin
  if btnLogPane.StyleLookup = 'buttonright' then OpenLogPane
  else CloseLogPane;
end;

//Animate left to right log pane
Procedure TfrmMain.OpenLogPane;
begin
  if btnLogPane.StyleLookup = 'buttonright' then
  FMX.ani.TAnimator.AnimateFloat(pnlLog, 'position.x', pnlLog.Position.X
+ pnlLog.Width, 1.5, TAnimationType.Out, TInterpolationType.Elastic);
  btnLogPane.StyleLookup := 'buttonleft';
end;

//Animate closing right to left of log pane
Procedure TfrmMain.CloseLogPane;
begin
  if btnLogPane.StyleLookup = 'buttonleft' then
  FMX.ani.TAnimator.AnimateFloat(pnlLog, 'position.x', 1 - pnlLog.Width,
1.5, TAnimationType.Out, TInterpolationType.Elastic);
  btnLogPane.StyleLookup := 'buttonright';
end;

//Toggle Minecraft Update Pane
Procedure TfrmMain.btnUpdatesPaneClick(Sender: TObject);
begin
 if btnUpdatesPane.StyleLookup = 'buttonleft' then OpenUpdatesPane
 else CloseUpdatesPane;
end;
```

```
//Open update pane left to right
Procedure TfrmMain.OpenUpdatesPane;
begin
  if btnUpdatesPane.StyleLookup = 'buttonleft' then
  FMX.ani.TAnimator.AnimateFloat(pnlUpdates, 'position.x', Width -
pnlUpdates.Width, 1.5, TAnimationType.Out, TInterpolationType.Elastic);
  btnUpdatesPane.StyleLookup := 'buttonright';
end;


//close updates pane right to left
Procedure TfrmMain.CloseUpdatesPane;
begin
  if btnUpdatesPane.StyleLookup = 'buttonright' then
  FMX.ani.TAnimator.AnimateFloat(pnlUpdates, 'position.x', Width - 1 ,
1.5, TAnimationType.Out, TInterpolationType.Elastic);
  btnUpdatesPane.StyleLookup := 'buttonleft';
end;


//Toggle Settings Pane
Procedure TfrmMain.btnSettingsPaneClick(Sender: TObject);
begin
 if btnSettingsPane.StyleLookup = 'buttonleft' then OpenSettingsPane
 else CloseSettingsPane;
end;


//Closing Settings Pane Left To Right
Procedure TfrmMain.CloseSettingsPane;
begin
  if btnSettingsPane.StyleLookup = 'buttonright' then
  FMX.ani.TAnimator.AnimateFloat(pnlSettings, 'position.x', Width - 1 ,
1.5, TAnimationType.Out, TInterpolationType.Elastic);
  btnSettingsPane.StyleLookup := 'buttonleft';
end;
```

```
//Opening Settings Pane Right to Left

Procedure TfrmMain.OpenSettingsPane;

begin

  if btnSettingsPane.StyleLookup = 'buttonleft' then

  FMX.ani.TAnimator.AnimateFloat(pnlSettings, 'position.x', Width -
pnlSettings.Width, 1.5, TAnimationType.Out, TInterpolationType.Elastic);

  btnSettingsPane.StyleLookup := 'buttonright';

end;


//Toggle Database Pane

Procedure TfrmMain.btnDatabasePaneClick(Sender: TObject);

begin

 if btnDatabasePane.StyleLookup = 'arrowdowntoolbutton' then
OpenDatabasePane

 else CloseDatabasePane;

end;


//Opening Database Pane Top to Bottom

Procedure TfrmMain.OpenDatabasePane;

begin

  if btnDatabasePane.StyleLookup = 'arrowdowntoolbutton' then

  FMX.ani.TAnimator.AnimateFloat(pnlDB, 'position.y', 0, 1.5,
TAnimationType.Out, TInterpolationType.Elastic);

  btnDatabasePane.StyleLookup := 'arrowuptoolbutton';

end;


//Closing Database Pane Bottom to Top

Procedure TfrmMain.CloseDatabasePane;

begin

  if btnDatabasePane.StyleLookup = 'arrowuptoolbutton' then

  FMX.ani.TAnimator.AnimateFloat(pnlDB, 'position.y', 1-pnlDB.Height ,
1.5, TAnimationType.Out, TInterpolationType.Elastic);

  btnDatabasePane.StyleLookup := 'arrowdowntoolbutton';

end;
```

```
//Toggle User Requests Pane

Procedure TfrmMain.btnRequestsUserPaneClick(Sender: TObject);

begin

  if btnRequestsUserPane.StyleLookup = 'buttonright' then
OpenRequestsUserPane

  else CloseRequestsUserPane;

end;


//Animate left to right user requests pane

Procedure TfrmMain.OpenRequestsUserPane;

begin

  if btnRequestsUserPane.StyleLookup = 'buttonright' then

  FMX.ani.TAnimator.AnimateFloat(pnlRequestUser, 'position.x',
pnlRequestUser.Position.X + pnlRequestUser.Width, 1.5,
TAnimationType.Out, TInterpolationType.Elastic);

  btnRequestsUserPane.StyleLookup := 'buttonleft';

end;


//Animate closing right to left of user requests pane

Procedure TfrmMain.CloseRequestsUserPane;

begin

  if btnRequestsUserPane.StyleLookup = 'buttonleft' then

  FMX.ani.TAnimator.AnimateFloat(pnlRequestUser, 'position.x', 1 -
pnlRequestUser.Width, 1.5, TAnimationType.Out,
TInterpolationType.Elastic);

  btnRequestsUserPane.StyleLookup := 'buttonright';

end;


//Toggle Sync Pane

Procedure TfrmMain.btnSyncPaneClick(Sender: TObject);

begin

  if btnSyncPane.StyleLookup = 'buttonright' then OpenSyncPane

  else CloseSyncPane;

end;


//Open the sync pane left to right
```

```
Procedure TfrmMain.OpenSyncPane;
begin
  if btnSyncPane.StyleLookup = 'buttonright' then
  FMX.ani.TAnimator.AnimateFloat(pnlSync, 'position.x',
pnlSync.Position.X + pnlSync.Width, 1.5, TAnimationType.Out,
TInterpolationType.Elastic);
  btnSyncPane.StyleLookup := 'buttonleft';
end;

//Close the sync pane right to left
Procedure TfrmMain.CloseSyncPane;
begin
  if btnSyncPane.StyleLookup = 'buttonleft' then
  FMX.ani.TAnimator.AnimateFloat(pnlSync, 'position.x', 1 -
pnlSync.Width, 1.5, TAnimationType.Out, TInterpolationType.Elastic);
  btnSyncPane.StyleLookup := 'buttonright';
end;
{$ENDREGION}

{$REGION 'Creation'}
Procedure TfrmMain.FormCreate(Sender: TObject);
begin
//Create logger
  Log := TLogger.Create;
  Log.OnLog := OnLog;
  Log.Info('HMC Started - Version ' + launcherVersion.toString);
  ZipFile := TZipFile.Create;
//Create Database Handler
  DB := TDBHandler.Create(Self);
  SettingManager := TSettingLoader.Create;
  Minecraft := TMinecraftInstance.Create;
  Username := CurrentUsername;
  DownloadManager := TDownloadManager.Create;
  DownloadManager.OnQueueEmpty := OnQueueEmpty;
  MinecraftUpdater := TMinecraftUpdater.Create;
```

```
  JSONGen := TJSONGenerator.Create;
  SyncManager := TFileSyncManager.Create;
end;

//Checks if the MySQL library is available, else extract it
Procedure TfrmMain.CreateMySQLLib;
var
  RStream : TResourceStream;
begin
  //Load MySQL lib if it doesn't exist
  if not FileExists('libmysql.dll') then
  begin
    RStream := TResourceStream.Create(MainInstance, 'libmysqlMS',
RT_RCDATA);
    RStream.SaveToFile('libmysql.dll');
    log.Warn('Installing libmysql.dll');
    RStream.Free;
  end;
end;

//A better way of calling Procedures after the form as been drawn
Procedure TfrmMain.FormActivate(Sender: TObject);
begin
  OnActivate := Nil;
  RandomiseBackground;
  CreateMySQLLib;
  SetHints;
  //Load settings from wherever, into the settings record. If all fails,
create a local settings file
  if FileExists('HMC.dat') then SettingManager.LoadFromFile('HMC.dat',
Settings)
  else
  try
    SettingManager.LoadFromExecutable(ParamStr(0), Settings);
    //If Xms or Xmx are loaded as 0, this means the settings file is not
valid and needs to be reset
```

```
    if (Settings.Xms = 0) or (Settings.Xmx = 0) then raise
Exception.Create('Settings are invalid');
  Except On E:Exception do
    begin
    //Set host to blank
      Settings.DBUrl := '';
    //Default exclusion regex
      Settings.ExcludeRegex := '.*\\saves\\.*' + #10 + '.*\\logs\\.*' +
#10 + '.*\\config\\.*';
    //Default RAM to 512MB and 2048MB
      Settings.Xms := 512;
      Settings.Xmx := 2048;
    //Default 30 seconds till termination
      Settings.SecondsTillTermination := 30;
    //If HMC was installed to the correct
MinectaftDir\launcher\standalone directory, automatically set the
Minecraft Directory
      if Pos('launcher\standalone',ParamStr(0)) > 0 then
Settings.CustomMinecraftDir := Copy(ParamStr(0), 1,
Pos('\launcher\standalone', ParamStr(0)));
      SettingManager.SaveToFile('HMC.dat', Settings);
    end;
  end;
  mmoRegex.Text := Settings.ExcludeRegex;

  if Length(Settings.copyTo) > 0 then
    cbxCopyTo.text := 'Automatically copy to ' + Settings.CopyTo
  else
    cbxCopyTo.IsChecked := False;
  if length(Settings.CustomMinecraftDir) > 0 then MinecraftDir :=
Settings.CustomMinecraftDir
  else MinecraftDir := RoamingAppDataPath + '\HMC';

  //If database is enabled, handle potential first launch, and set up
settings. Otherwise assume admin
  if Length(Settings.DBUrl) > 0 then
  begin
```

```
    DB.SetDatabase(Settings.DBUrl, Settings.DBUserName,
Settings.DBPassword);
    if DB.DBServerExists then
      if DB.DBExists then
      begin
        if not DB.GetUserID(Username) then
        begin
          DB.CreateUser(Username);
          DB.GetUserID(UserName);
        end;
        Settings.Alias := DB.GetAlias;
        isAdmin := DB.isAdmin;
        if isAdmin then
        begin
          edtName.ReadOnly := False;
          cbxDB.IsChecked := True;
          edtDBURL.Text := Settings.DBURL;
          edtDBUsername.Text := Settings.DBUsername;
          edtDBPassword.Text := Settings.DBPassword;
          edtDBUsername.Enabled := True;
          edtDBPassword.Enabled := True;
          btnInitialiseDB.Enabled := True;
          btnApplyDB.Enabled := True;
          btnRefreshClick(Self);
          pnlDB.Visible := True;
          pnlSync.Visible := True;
        end
        else
        begin
        //Not an admin, so enable student features
          edtName.ReadOnly := True;
          pnlSettings.Visible := False;
          pnlDB.Visible := False;
          pnlSync.Visible := False;
```

```
        DB.OnQuotaChange := OnQuotaChange;
        DB.OnQuotaUp := OnQuotaUp;
        DB.OnBanned := OnBan;
        DB.EnableBanChecker;
        lblQuota.Visible := True;
        lblQuota.Text := DB.GetQuota.ToString + ' minutes remaining';
        if DB.GetQuota = 0 then btnPlay.Visible := False;
      end;
    end
    else
    begin
      DB.InitialiseDB;
      DB.CreateUser(UserName);
      DB.GetUserID(UserName);
      DB.SetAdmin(DB.LocalUserID);
      Self.FormActivate(Self);
    end
  else Log.Warn('Unable to connect to Database');
end
else isAdmin := True;

//If isAdmin regardless of database settings. Toggle visual settings
if isAdmin then
begin
  pnlSettings.Visible := True;
  pnlRequestUser.Visible := False;
  pnlSync.Visible := True;
  pnlSettings.Visible := True;
  pnlUpdates.Visible := True;
  edtName.ReadOnly := False;
  edtName.OnChange := edtNameChange;
  edtName.Text := Settings.Alias;
  sbxQuotaTTT.Value := Settings.SecondsTillTermination;
  edtMCDir.Text := Settings.CustomMinecraftDir;
```

```
    edtFileRepo.Text := Settings.FileRepoURL;
    //Prepare Minecraft Updater
    MinecraftUpdater.OnNewDownload := OnNewDownload;
    MinecraftUpdater.OnGetVersions := OnGetVersionsList;
    MinecraftUpdater.MinecraftDir := MinecraftDir;
    if InternetConnected then MinecraftUpdater.GetVersions;
  end;
  mmoRegex.Text := Settings.ExcludeRegex;
  //Admin or user
  if Length(Settings.Alias) > 0 then edtName.Text := Settings.Alias
  else edtName.Text := Username;
  if GetJavaExe = '' then
  begin
    Log.Error('The Java Runtime Environment was not detected.');
    Log.Error('Please ensure the 32/64bit version of JRE is installed as
appropriate');
    btnPlay.Visible := False;
  end;

  //If there is a file repository url and user is connected to internet,
sync the files
  if (Length(Settings.FileRepoURL) > 0) and (InternetConnected) then
  begin
    SyncManager.MinecraftDir := MinecraftDir;
    SyncManager.RootURL := Settings.FileRepoURL;
    SyncManager.OnNewDownload := OnNewDownload;
    SyncManager.OnUpdateLauncher := OnUpdateLauncher;
    //Only synchronise if this is the master copy
    if not((isAdmin) and (Pos('standalone', ParamStr(0)) > 0)) then
      SyncManager.Compare;
    btnPlay.Enabled := False;
    aniBusy.Enabled := True;
  end;
end;
```

```
{$ENDREGION}


{$REGION 'Destruction'}
Procedure TfrmMain.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Minecraft.Destroy;
  if FileExists('HMC.dat') then SettingManager.SaveToFile('HMC.dat',
Settings);
  Log.SaveToFile('log.txt');
  SettingManager.Destroy;
  JSONGen.Destroy;
  SyncManager.Destroy;
end;


Procedure TfrmMain.FormDeactivate(Sender: TObject);
begin
  RandomiseBackground;
end;
{$ENDREGION}


{$REGION 'Grids'}
{$REGION 'Popup Menus'}
{$REGION 'Requests'}
//Requests Popup Menu
Procedure TfrmMain.PopupMenuRequestsPopup(Sender: TObject);
var
  LocalMousePosition : TPointF;
begin
  LocalMousePosition := SgdRequests.AbsoluteToLocal(Screen.MousePos);
  SgdRequests.Selected := SgdRequests.RowByPoint(LocalMousePosition.X,
LocalMousePosition.Y) - 1;
  if SgdRequests.RowCount > 0 then

    if SgdRequests.Cells[2,SgdRequests.Selected] = 'Alias' then
    begin
```

```
      MimAcceptAlias.Visible := True;
      MimDeclineAlias.Visible := True;
      MimProcessRequest.Visible := False;
    end
    else
    begin
      MimAcceptAlias.Visible := False;
      MimDeclineAlias.Visible := False;
      MimProcessRequest.Visible := True;
    end;
end;

//Sets alias of selected user
Procedure TfrmMain.mimAcceptAliasClick(Sender: TObject);
begin
  //Set the Alias
  DB.SetAlias(SgdRequests.Cells[5,SgdRequests.Selected].ToInteger,
SgdRequests.Cells[3,SgdRequests.Selected]);
  //Hide it
  DB.ProcessRequest(SgdRequests.Cells[4,SgdRequests.Selected].ToInteger);
  btnRefreshClick(Sender);
end;

//Ignores request
Procedure TfrmMain.mimDeclineAliasClick(Sender: TObject);
begin
  //Hide it
  DB.ProcessRequest(SgdRequests.Cells[4,SgdRequests.Selected].ToInteger);
  btnRefreshClick(Sender);
end;

//Processes request
Procedure TfrmMain.MimProcessRequestClick(Sender: TObject);
begin
```

```
  //Hide it
  DB.ProcessRequest(SgdRequests.Cells[4,SgdRequests.Selected].toInteger);
  btnRefreshClick(Sender);
end;

{$ENDREGION}

{$REGION 'Users'}
//Forces right-clicked row to be selected
Procedure TfrmMain.PopupMenuDBPopup(Sender: TObject);
var
  LocalMousePosition : TPointF;
begin
  //To Stop erroring
  if SgdUsers.RowCount = 0 then SgdUsers.RowCount := 1;
  LocalMousePosition := SgdUsers.AbsoluteToLocal(Screen.MousePos);
  SgdUsers.Selected := SgdUsers.RowByPoint(LocalMousePosition.X,
LocalMousePosition.Y) - 1;
end;

Procedure TfrmMain.mimAliasClick(Sender: TObject);
Var
  NewAlias : String;
begin
  //First check that there is something in that row
  if SgdUsers.Cells[0, SgdUsers.Selected].Length > 0 then
  begin
    //Request alias from user
    NewAlias := InputBox('Change Alias', 'Enter a New Alias', '');
    if NewAlias.Length > 0 then DB.SetAlias(SgdUsers.Cells[7,
SgdUsers.Selected].ToInteger, NewAlias);
    btnRefreshClick(Self);
  end;
end;
```

```
//Ban selected user
Procedure TfrmMain.mimBanClick(Sender: TObject);
begin
  //First check that there is something in that row
  if SgdUsers.Cells[0, SgdUsers.Selected].Length > 0 then
  begin
    //Ban
    DB.SetBanned(SgdUsers.Cells[7, SgdUsers.Selected].ToInteger);
    btnRefreshClick(Self);
  end;
end;

Procedure TfrmMain.mimGlobalQuotaClick(Sender: TObject);
Var
  QuotaStr : String;
  QuotaInt : Integer;
begin
  //First check that there is something in that row
  if SgdUsers.Cells[0, SgdUsers.Selected].Length > 0 then
  begin
    //Request alias from user
    QuotaStr := InputBox('Change Global Quota', 'New Global Quota', '');
    if TryStrToInt(QuotaStr, QuotaInt) then
      if QuotaInt >= 0 then
        DB.ChangeDefaultQuota(QuotaInt)
      else Showmessage('Please enter an integer')
    else Showmessage('Please enter a positive integer');
    btnRefreshClick(Self);
  end;
end;

Procedure TfrmMain.mimQuotaClick(Sender: TObject);
Var
  QuotaStr : String;
```

```
  QuotaInt : Integer;
begin
  //First check that there is something in that row
  if SgdUsers.Cells[0, SgdUsers.Selected].Length > 0 then
  begin
    //Request alias from user
    QuotaStr := InputBox('Change User Quota', 'New User Quota', '');
    if TryStrToInt(QuotaStr, QuotaInt) then
      if QuotaInt >= 0 then
        DB.SetQuota(SgdUsers.Cells[7, SgdUsers.Selected].ToInteger,
QuotaInt)
        else Showmessage('Please enter an integer')
      else Showmessage('Please enter a positive integer');
    btnRefreshClick(Self);
  end;
end;

//Revoke admin rights from selected user
Procedure TfrmMain.mimRevokeAdminClick(Sender: TObject);
begin
  //First check that there is something in that row
  if SgdUsers.Cells[0, SgdUsers.Selected].Length > 0 then
  begin
    DB.RevokeAdmin(SgdUsers.Cells[7, SgdUsers.Selected].ToInteger);
    btnRefreshClick(Self);
  end;
end;

//Set Admin for selected user
Procedure TfrmMain.mimSetAdminClick(Sender: TObject);
begin
  //First check that there is something in that row
  if SgdUsers.Cells[0, SgdUsers.Selected].Length > 0 then
  begin
```

```
    DB.SetAdmin(SgdUsers.Cells[7, SgdUsers.Selected].ToInteger);
    btnRefreshClick(Self);
  end;
end;

//Unban selected user
Procedure TfrmMain.mimUnbanClick(Sender: TObject);
begin
  //First check that there is something in that row
  if SgdUsers.Cells[0, SgdUsers.Selected].Length > 0 then
  begin
    DB.SetUnbanned(SgdUsers.Cells[7, SgdUsers.Selected].ToInteger);
    btnRefreshClick(Self);
  end;
end;

{$ENDREGION}
{$ENDREGION}

{$REGION 'Download Grid'}
//Updates grid values
Procedure TfrmMain.OnDownloadProgress(Sender: TObject; URL: string;
Progress: Integer);
var
  i : integer;
begin
//If any row matches the URL, update its progress
  for i := Low(Downloads) to High(Downloads) do if Downloads[i] = URL
then
  begin
    DProgress[i] := Progress;
    grdDownloads.Columns[0].UpdateCell(i);
    grdDownloads.Columns[1].UpdateCell(i);
    Exit;
  end;
```

```
end;


//Remove values from grid when download finishes
Procedure TfrmMain.OnDownloadFinish(Sender: TObject; URL: string);
var
  i : integer;
begin
  for i := Low(Downloads) to High(Downloads) do if Downloads[i] = URL
then
  begin
    Downloads[i] := '';
    DProgress[i] := 0;
    grdDownloads.Columns[0].UpdateCell(i);
    grdDownloads.Columns[1].UpdateCell(i);
    Exit;
  end;
end;


//Updates the grid when an actual download begins
Procedure TfrmMain.OnDownloadBegin(Sender: TObject; URL: string);
var
  i : Integer;
begin
  //If there is no URL in the row then use this row
  for i := Low(Downloads) to High(Downloads) do
    if Downloads[i] = '' then
    begin
      Downloads[i] := URL;
      grdDownloads.Columns[0].UpdateCell(i);
      grdDownloads.Columns[1].UpdateCell(i);
      btnPlay.Enabled := False;
      aniBusy.Visible := True;
      Exit;
    end;
```

```
end;


//When the queue is empty, disable the spinner and enable play button.
This event will be fired multiple times
Procedure TfrmMain.OnQueueEmpty(Sender: TObject);
begin
  aniBusy.Visible := False;
  btnPlay.Enabled := True;
end;


//Values that should fill in grid when UpdateColumns is called
Procedure TfrmMain.grdDownloadsGetValue(Sender: TObject; const Col,
  Row: Integer; var Value: TValue);
begin
  if Col = 0 then Value := TValue.From<string>(Downloads[Row])
  else if Col = 1 then Value := TValue.From<integer>(DProgress[Row]);
end;


//Wrapper method to make calling download less bloated
Procedure TfrmMain.AddDownload(URL: string; FilePath: string; OnComplete
: TDownloadEvent);
begin
  DownloadManager.AddDownload(URL, FilePath, OnDownloadBegin,
OnDownloadProgress, OnDownloadFinish, nil, OnComplete);
end;


//Wrapper callback for assigning
Procedure TfrmMain.OnNewDownload(Sender: TObject; URL: string; FilePath:
string; OnComplete: TDownloadEvent);
begin
  AddDownload(URL, FilePath, OnComplete);
end;
{$ENDREGION}


{$REGION 'Database Grids'}
//Refresh based on active tab
```

```
Procedure TfrmMain.btnRefreshClick(Sender: TObject);
begin
  if tclDB.ActiveTab.Text = 'Users' then timUsersClick(Sender)
  else if tclDB.ActiveTab.Text = 'Sessions' then timSessionsClick(Sender)
  else if tclDB.ActiveTab.Text = 'Messages' then timMessagesClick(Sender)
  else if tclDB.ActiveTab.Text = 'Requests' then
timRequestsClick(Sender);
end;

//Search the current grid characters are entered
Procedure TfrmMain.edtSearchChangeTracking(Sender: TObject);
begin
  if tclDB.ActiveTab.Text = 'Users' then
  begin
    SgdUsers.RowCount := 0;
    DB.SearchUser(edtSearch.Text);
  end
  else if tclDB.ActiveTab.Text = 'Sessions' then
  begin
    SgdSessions.RowCount := 0;
    DB.SearchSessions(edtSearch.Text);
  end
  else if tclDB.ActiveTab.Text = 'Messages' then
  begin
    SgdMessages.RowCount := 0;
    DB.SearchMessages(edtSearch.Text);
  end
  else if tclDB.ActiveTab.Text = 'Requests' then
  begin
    SgdRequests.RowCount := 0;
    DB.SearchRequests(edtSearch.Text);
  end;
end;
```

```
//Get messages
Procedure TfrmMain.timMessagesClick(Sender: TObject);
begin
  sgdMessages.RowCount := 0;
  edtSearch.Visible := True;
  DB.OnRowFetch := PopulateMessages;
  DB.GetMessages;
end;

//Get requests
Procedure TfrmMain.timRequestsClick(Sender: TObject);
begin
  sgdRequests.RowCount := 0;
  edtSearch.Visible := True;
  DB.OnRowFetch := PopulateRequests;
  DB.GetRequests;
end;

//Get sessions
Procedure TfrmMain.timSessionsClick(Sender: TObject);
begin
  sgdSessions.RowCount := 0;
  edtSearch.Visible := True;
  DB.OnRowFetch := PopulateSessions;
  DB.GetSessions;
end;

//Refresh view and enable search
Procedure TfrmMain.timUsersClick(Sender: TObject);
begin
  sgdUsers.RowCount := 0;
  edtSearch.Visible := True;
  DB.OnRowFetch := PopulateUsers;
  DB.GetUsers;
```

```
end;


//Populates the grid row by row
Procedure TfrmMain.PopulateUsers(Sender: TObject; Fields : TFields);
begin
  sgdUsers.RowCount := sgdUsers.RowCount + 1;
  sgdUsers.Cells[0,sgdUsers.RowCount - 1] :=
Fields.FieldByName('LoginName').AsString;
  sgdUsers.Cells[1,sgdUsers.RowCount - 1] :=
Fields.FieldByName('Alias').AsString;
  sgdUsers.Cells[2,sgdUsers.RowCount - 1] :=
Fields.FieldByName('Banned').AsString;
  sgdUsers.Cells[3,sgdUsers.RowCount - 1] :=
Fields.FieldByName('Admin').AsString;
  sgdUsers.Cells[4,sgdUsers.RowCount - 1] :=
Round(Fields.FieldByName('Average').AsSingle).ToString + ' minutes';
  sgdUsers.Cells[5,sgdUsers.RowCount - 1] :=
Fields.FieldByName('SessionCount').AsString;
  sgdUsers.Cells[6,sgdUsers.RowCount - 1] :=
Fields.FieldByName('Quota').AsString + ' minutes';
  sgdUsers.Cells[7,sgdUsers.RowCount - 1] :=
Fields.FieldByName('UserID').AsString;
end;


//Populates the grid row by row
Procedure TfrmMain.PopulateSessions(Sender: TObject; Fields : TFields);
begin
  sgdSessions.RowCount := sgdSessions.RowCount + 1;
  sgdSessions.Cells[0,sgdSessions.RowCount - 1] :=
Fields.FieldByName('LoginName').AsString;
  sgdSessions.Cells[1,sgdSessions.RowCount - 1] :=
Fields.FieldByName('LogOnTime').AsString;
  sgdSessions.Cells[2,sgdSessions.RowCount - 1] :=
Fields.FieldByName('LogOffTime').AsString;
  sgdSessions.Cells[3,sgdSessions.RowCount - 1] :=
Fields.FieldByName('ComputerName').AsString;
end;


//Populates the grid row by row with messages
```

```
Procedure TfrmMain.PopulateMessages(Sender: TObject; Fields : TFields);
begin
  sgdMessages.RowCount := sgdMessages.RowCount + 1;
  sgdMessages.Cells[0,sgdMessages.RowCount - 1] :=
Fields.FieldByName('LoginName').AsString;
  sgdMessages.Cells[1,sgdMessages.RowCount - 1] :=
Fields.FieldByName('TimeSent').AsString;
  sgdMessages.Cells[2,sgdMessages.RowCount - 1] :=
Fields.FieldByName('MessageText').AsString;
end;

//Populates grid with requests
Procedure TfrmMain.PopulateRequests(Sender : TObject; Fields : TFields);
begin
  sgdRequests.RowCount := sgdRequests.RowCount + 1;
  sgdRequests.Cells[0,sgdRequests.RowCount - 1] :=
Fields.FieldByName('LoginName').AsString;
  sgdRequests.Cells[1,sgdRequests.RowCount - 1] :=
Fields.FieldByName('RequestedDate').AsString;
  sgdRequests.Cells[2,sgdRequests.RowCount - 1] :=
Fields.FieldByName('RequestType').AsString;
  sgdRequests.Cells[3,sgdRequests.RowCount - 1] :=
Fields.FieldByName('RequestValue').AsString;
  sgdRequests.Cells[4,sgdRequests.RowCount - 1] :=
Fields.FieldByName('RequestID').AsString;
  sgdRequests.Cells[5,sgdRequests.RowCount - 1] :=
Fields.FieldByName('UserID').AsString;
end;



{$ENDREGION}

{$REGION 'Logger Grid'}
//When something logs to the logger, add it to the grid
Procedure TfrmMain.OnLog(Text: string; EntryType: TEntryType);
begin
  grdLog.RowCount := grdLog.RowCount + 1;
  grdLog.Cells[0, grdLog.RowCount - 1] := Text;
```

```
  if EntryType = TEntryType.Error then openLogPane;
end;


//Colour in row red or orange if error or warning
Procedure TfrmMain.grdLogDrawColumnCell(Sender: TObject; const Canvas:
TCanvas;
  const Column: TColumn; const Bounds: TRectF; const Row: Integer;
  const Value: TValue; const State: TGridDrawStates);
var
  RowColour : TBrush;
begin
  RowColour := TBrush.Create(TBrushKind.Solid, TAlphaColors.Null);
  //If there is an error
  if Pos('[Error]', Value.ToString) > 0 then RowColour.Color :=
TAlphaColors.Red
  else if Pos('[Warning]', Value.ToString) > 0 then RowColour.Color :=
TAlphaColors.Orange;
  Canvas.FillRect(Bounds, 0, 0, [], 0.4, RowColour);
  TGrid(Sender).DefaultDrawColumnCell(Canvas, Column, Bounds, Row, Value,
State);
  RowColour.Free;
end;
{$ENDREGION}
{$ENDREGION}


{$REGION 'Database Events'}
//If the player is banned at any time, they will be sure of it
Procedure TfrmMain.OnBan(Sender: TObject);
begin
  txtBanned.Visible := True;
  btnPlay.Visible := False;
  Minecraft.Close;
end;


//Updates the quota overlay
Procedure TfrmMain.OnQuotaChange(Sender: TObject; MinutesLeft: Integer);
```

```
begin
  if MinutesLeft > 1 then Minecraft.SetMessageBox(MinutesLeft.ToString +
' Minutes Remaining')
  else Minecraft.SetMessageBox('1 Minute Remaining');
end;


//Starts the termination count down
Procedure TfrmMain.OnQuotaUp(Sender: TObject);
begin
  btnPlay.Enabled := False;
  lblQuota.Text := 'Time Up';
  Minecraft.SetLavaOverlay;
  Minecraft.TerminateIn(settings.SecondsTillTermination);
end;
{$ENDREGION}


{$REGION 'Settings Synchronisation/Validation'}
//Set hints for all of the buttons
Procedure TfrmMain.SetHints;
begin
//Set the Hint colour and hints for all the buttons
  SetHintSetting(1, 10000, True, TAlphaColors.Deepskyblue);
  SetAHint(btnDatabasePane, 'Database Pane - Manage Users');
  SetAHint(btnSyncPane, 'Sync Pane - Synchronise Mods and Exclude
Files');
  SetAHint(btnLogPane, 'Log Pane');
  SetAHint(btnSettingsPane, 'Settings Pane - Configure RAM, Backups,
Student Settings and Database Settings');
  SetAHint(btnRequestsUserPane, 'Requests Pane - Ask for a new Username
or Mod');
  SetAHint(btnUpdatesPane, 'Minecraft Install Pane - Install Different
Minecraft Versions');
end;


//Exclude regex
Procedure TfrmMain.mmoRegexChange(Sender: TObject);
```

```
begin
  Settings.ExcludeRegex := Trim(mmoRegex.Text);
end;

//TTT in seconds
Procedure TfrmMain.sbxQuotaTTTChange(Sender: TObject);
begin
  Settings.SecondsTillTermination :=
sbxQuotaTTT.Value.ToString.ToInteger;
end;

//Update values of Xms
Procedure TfrmMain.ttbXmsTracking(Sender: TObject);
begin
  lblXms.Text := ttbXms.Value.ToString;
  Settings.Xms := ttbXms.Value.ToString.ToInteger;
end;

//Update Xmx values
Procedure TfrmMain.ttbXmxTracking(Sender: TObject);
begin
  lblXmx.Text := ttbXmx.Value.ToString;
  Settings.Xmx := ttbXmx.Value.ToString.ToInteger;
end;

//Ensure that Xms is always less than Xmx
Procedure TfrmMain.ttbXmxChange(Sender: TObject);
begin
  if ttbXms.Value > ttbXmx.Value then
  begin
    ttbXms.Value := 256;
    showmessage('You cannot have a larger Starting RAM than Maximum
RAM.');
  end;
end;
```

```
//Reset Java Xms and Xmx
Procedure TfrmMain.btnDefaultJavaClick(Sender: TObject);
begin
  ttbXms.Value := 512;
  ttbXmx.Value := 2048;
end;


//Updates settings and if need be the database if an admin changes their
alias
Procedure TfrmMain.ebnDBPasswordClick(Sender: TObject);
begin
  edtDBPassword.Password := not edtDBPassword.Password;
end;


//Check that repository is contactable
Procedure TfrmMain.edtFileRepoChange(Sender: TObject);
begin
  if edtFileRepo.Text.Length > 100 then
  begin
    edtFileRepo.Text := '';
    Showmessage('The URL must be 100 or less characters');
  end;
  if edtFileRepo.Text.Length > 0 then
    if pos('http', edtFileRepo.Text) = 0 then
    begin
      edtFileRepo.Text := '';
      Showmessage('The URL must be in the form of http://server/path');
    end;
  //Strip the slash
  if edtFileRepo.Text.Length > 0 then
    if edtFileRepo.Text[edtFileRepo.Text.Length] = '/' then
edtFileRepo.Text := Copy(edtFileRepo.Text, 1, edtFileRepo.Text.Length -
1);
```

```
  Settings.FileRepoURL := edtFileRepo.Text;
end;


//Validates Minecraft Directory
Procedure TfrmMain.edtMCDirChange(Sender: TObject);
begin
  if (edtMCDir.Text.Length > 100) or (edtMCDir.Text.IsEmpty) then
  begin
    edtMCDir.Text := '';
    MinecraftDir := RoamingAppdataPath + '\HMC';
    if edtMCDir.Text.Length > 100 then Showmessage('The Directory must be
100 or less characters long.');
  end
  else
  begin
    Settings.CustomMinecraftDir := edtMCDir.Text;
    MinecraftDir := Settings.CustomMinecraftDir;
  end;
   //Strip the slash
  if edtMCDir.Text.Length > 0 then
    if edtMCDir.Text[edtMCDir.Text.Length] = '\' then edtMCDir.Text :=
Copy(edtMCDir.Text, 1, edtMCDir.Text.Length - 1);
  Settings.CustomMinecraftDir := MinecraftDir;
  Log.Info('Minecraft Directory Set to: ' + MinecraftDir);
end;


//Validates custom alias for admin
Procedure TfrmMain.edtNameChange(Sender: TObject);
begin
  if (edtName.Text.Length > 20) OR (edtName.Text.Length = 0) then
  begin
    edtName.Text := Username;
    Showmessage('Please enter a username that is between 1-20
characters');
  end;
```

```
  Settings.Alias := edtName.Text;
  if Length(Settings.DBUrl) > 0 then DB.SetAlias(DB.LocalUserID,
edtName.Text);
end;

//Modifies the CopyTo field
Procedure TfrmMain.cbxCopyToChange(Sender: TObject);
begin
  if cbxCopyTo.IsChecked then
    if OpenDialogSync.Execute then
    begin
      Settings.copyTo := ExtractFilePath(OpenDialogSync.FileName);
      cbxCopyTo.Text := 'Automatically Copy To ' + Settings.CopyTo;
    end
    else
    begin
      Settings.CopyTo := '';
      cbxCopyTo.IsChecked := False;
      cbxCopyTo.Text := 'Automatically Copy To';
    end
  else
  begin
    Settings.CopyTo := '';
    cbxCopyTo.IsChecked := False;
    cbxCopyTo.Text := 'Automatically Copy';
  end;
end;

//Sets up and validates the database connection
Procedure TfrmMain.btnApplyDBClick(Sender: TObject);
var
  ValidationSuccess : Boolean;
begin
  ValidationSuccess := False;
```

```
  //Check all are within limits
  if (edtDBURL.Text.Length > 0) AND (edtDBURL.Text.Length <= 100) then
    if edtDBUsername.Text.Length <= 20 then
      if edtDBPassword.Text.Length <= 20 then ValidationSuccess := True
      else Showmessage('Password must be 20 or less characters')
    else Showmessage('Username must be 20 or less characters')
  else Showmessage('Database URL must be between 1 and 100 characters');
  //Only save details and check connection if this is all true
  if ValidationSuccess then
    begin
      DB.SetDatabase(edtDBURL.Text, edtDBUsername.Text,
edtDBPassword.Text);
      if DB.DBServerExists then
        if DB.DBExists then
        begin
          Log.Info('Connection to Database established');
          Showmessage('Connection to Database Establised');
          Settings.DBUrl := edtDBURL.Text;
          Settings.DBUserName := edtDBUsername.Text;
          Settings.DBPassword := edtDBPassword.Text;
          btnInitialiseDB.Enabled := True;
          if not DB.GetUserID(UserName)
          then
          begin
            DB.CreateUser(UserName);
            DB.GetUserID(UserName);
          end;
          DB.SetAdmin(DB.LocalUserID);
          pnlDB.Enabled := True;
        end
        else
        begin
          DB.InitialiseDB;
          Log.Warn('Connection to Database established. No Database
found. Initialising');
```

```
          Showmessage('Connection to Database Estabilised. No Database
Found. Will Automatically Prepare the Database');
          DB.CreateUser(UserName);
          DB.GetUserID(UserName);
          DB.SetAdmin(DB.LocalUserID);
      end
      else
      begin
        showmessage('Unable to connect to the database. Please check your
settings and try again.');
        pnlDB.Enabled := False;
      end;
    end;
end;


//Creates user executable. No overwriting the running file
Procedure TfrmMain.btnGenerateExecutableClick(Sender: TObject);
begin
  if SaveExeDialog.Execute then
    if not (SaveExeDialog.FileName = ParamStr(0))
      then SettingManager.GenerateLauncher(SaveExeDialog.FileName,
Settings)
    else Showmessage('You cannot overwrite the currently running
executable. Please select a different file path.');
end;


//Confirm the initialisation of the database
Procedure TfrmMain.btnInitialiseDBClick(Sender: TObject);
begin
  if MessageDlg('Are you sure you want to initialise the HMC database?
This will delete everything in it.',TMsgDlgType.mtConfirmation,
[TMsgDlgBtn.mbYes, TMsgDlgBtn.mbNo], 0) = mrYes
  then
  begin
    DB.InitialiseDB;
    DB.CreateUser(UserName);
```

```
    DB.GetUserID(UserName);
    DB.SetAdmin(DB.LocalUserID);
  end;
end;

//Disable DB in the record, and disable initialising
Procedure TfrmMain.cbxDBChange(Sender: TObject);
begin
  if not cbxDB.isChecked then
  begin
    Settings.DBUrl := '';
    btnInitialiseDB.Enabled := False;
  end;
end;
{$ENDREGION}

{$REGION 'Backup Management'}
//Saves backup as zip file
Procedure TfrmMain.btnSaveBackupClick(Sender: TObject);
begin
  if SaveBackupDialog.Execute then
  try
    //Do not ZIP exe as exe or it will fail on extraction
    if FileExists(ExtractFilePath(paramStr(0)) + 'HMC.old') then
      TFile.Delete(ExtractFilePath(paramStr(0)) + 'HMC.old');
    RenameFile(ParamStr(0), ExtractFilePath(paramStr(0)) + 'HMC.old');
    if FileExists(ExtractFilePath(paramStr(0)) + 'libmysql.dll.old') then
      TFile.Delete(ExtractFilePath(paramStr(0)) + 'libmysqll.dll.old');
    RenameFile(ExtractFilePath(paramStr(0)) + 'libmysqll.dll',
ExtractFilePath(paramStr(0)) + 'libmysqll.dll.old');
    ZipFile.ZipDirectoryContents(SaveBackupDialog.FileName,
MinecraftDir);
    RenameFile(ExtractFilePath(paramStr(0)) + 'HMC.old', ParamStr(0));
    RenameFile(ExtractFilePath(paramStr(0)) + 'libmysqll.dll.old',
ExtractFilePath(paramStr(0)) + 'libmysqll.dll');
```

```
    Log.Info('Created Backup');
  Except On E:Exception do Log.Error('Unable to Zip file: ' + E.Message);
  end;
end;

//Loads backup as zip file
Procedure TfrmMain.btnLoadBackupClick(Sender: TObject);
begin
  if OpenBackupDialog.Execute then if
ZipFile.IsValid(OpenBackupDialog.FileName) then
  try
    ZipFile.ExtractZipFile(OpenBackupDialog.FileName, MinecraftDir);
    Log.Info('Restored Backup');
    Showmessage('Backup Restored Sucessfully');
  Except On E:Exception do Log.Error('Unable to extract file: ' +
E.Message);
  end
  else Log.Error('Not a Valid Backup Archive');
end;

{$ENDREGION}

{$REGION 'Minecraft Instance'}
Procedure TfrmMain.btnPlayClick(Sender: TObject);
begin
  //Quick check to see if Minecraft has been synced yet
  if FileExists(MinecraftDir + '\launcher_profiles.json') then
  begin
    //If database mode is enabled
    if DB.DBServerExists then
    begin
      Minecraft.onSendMessage := OnMessageSend;
      Minecraft.OnClose := OnMinecraftClose;
      DB.BeginSession(ComputerName);
      //And student
```

```
      if not IsAdmin then
      begin
        DB.EnableQuota;
        Minecraft.Launch(edtName.Text, Settings.Xms, Settings.Xmx,
MinecraftDir);
        Minecraft.SetMessageBox(DB.GetQuota.ToString + ' Minutes
Remaining');
        WindowState := TWindowState.wsMinimized;
      end else Minecraft.Launch(edtName.Text, Settings.Xms, Settings.Xmx,
MinecraftDir);
    end
    else
    begin
      Minecraft.Launch(edtName.Text, Settings.Xms, Settings.Xmx,
MinecraftDir);
      WindowState := TWindowState.wsMinimized;
    end;
  end
  else Log.Error('Unable to launch. Please ensure Minecraft is
installed');
end;

//Insert Message into Database if one is detected
Procedure TfrmMain.OnMessageSend(Sender: TObject; Text: string);
begin
  DB.InsertMessage(Text);
end;

//End database session if Minecraft closes
Procedure TfrmMain.OnMinecraftClose(Sender: TObject);
begin
  DB.EndSession;
  //Close automatically if the user is not an administrator
  if not isAdmin then Close;
end;
{$ENDREGION}
```

```
{$REGION 'User Request'}
//Ensures entered request is no more than 20 characters
Procedure TfrmMain.edtNewAliasChange(Sender: TObject);
begin
  if edtNewAlias.text.Length > 20 then
  begin
    edtNewAlias.Text := '';
    Showmessage('Alias must be less than 20 characters');
  end;
end;

//Send the request
Procedure TfrmMain.btnSendRequestClick(Sender: TObject);
begin
  if tclUserRequest.ActiveTab.Text = 'Request a New Alias' then
  begin
    if edtNewAlias.Text.Length > 0 then
    begin
      DB.RequestAlias(edtNewAlias.Text);
      Showmessage('Request Submitted!');
    end
    else Showmessage('You must enter an alias!')
  end
    else if edtModRequest.Text.Length > 0 then
    begin
      DB.RequestMod(edtModRequest.Text + '. ' + mmoBecause.Text);
      Showmessage('Request Submitted!');
    end
    else Showmessage('You must enter a Mod name!');
end;

{$ENDREGION}
```

```
{$REGION 'Update Minecraft Version'}
//Populate combobox with list
Procedure TfrmMain.OnGetVersionsList(Sender: TObject; List: TStringList);
begin
  //Make a value copy of list items into combobox
  cmbVersions.Items.Assign(List);
  cmbVersions.ItemIndex := 0;
  btnInstall.Enabled := True;
end;

//Triggers the installing of Minecraft
Procedure TfrmMain.btnInstallClick(Sender: TObject);
begin
  MinecraftUpdater.Install(cmbVersions.Selected.Text);
  btnInstall.Enabled := False;
end;
{$ENDREGION}

{$REGION 'Mod Synchronization'}
//Saves all files to JSON
Procedure TfrmMain.btnSyncModsClick(Sender: TObject);
begin
  //Copying current application and student application to /launcher/. Create dirs first
  if not DirectoryExists(MinecraftDir + '\launcher\user') then ForceDirectories(MinecraftDir + '\launcher\user');
  if not DirectoryExists(MinecraftDir + '\launcher\standalone') then ForceDirectories(MinecraftDir + '\launcher\standalone');
  SettingManager.GenerateLauncher(MinecraftDir + '\launcher\user\HMC.exe', Settings);
  //Copy current file to \launcher\standalone\HMC.exe if it is running from somewhere else
  if not (ExtractFilePath(ParamStr(0)) = MinecraftDir + '\launcher\standalone\') then
    TFile.Copy(ParamStr(0), MinecraftDir + '\launcher\standalone\HMC' + ExtractFileExt(ParamStr(0)), True);
  Log.Info('Generating Files.JSON');
```

```
  JSONGen.Generate(MinecraftDir, mmoRegex.Lines);
  JSONGen.SaveTo(MinecraftDir + '\Files.json');
  if Length(Settings.CopyTo) > 0 then
  begin
    Log.Info('Copying to ' + Settings.CopyTo);
    try
      if DirectoryExists(Settings.CopyTo) then
        TDirectory.Delete(Settings.CopyTo, True);
      ForceDirectories(Settings.CopyTo);
      TDirectory.Copy(MinecraftDir, Settings.CopyTo);
    Except On E:Exception do
    begin
      Log.Error('Could not automatically copy files');
      Log.Error(E.ClassName + ':' + E.Message);
    end;
    end;
  end;
end;

//Close launcher and open new version
Procedure TfrmMain.OnUpdateDownloaded(Sender: TObject; URL: String);
begin
  Showmessage('Launcher updated, and will close. Please re-open the
launcher.');
  Close;
end;

//If there is a launcher update, deal with it here
Procedure TfrmMain.OnUpdateLauncher(Sender: TObject);
begin
  try
    log.Warn('There is a launcher update available');
  //If the current exe is the same as
MinecraftDir\launcher\standalone\HMC.exe or user\HMC.exe depending on
permissions
```

```
    if isAdmin then
      if ParamStr(0) = MinecraftDir + '\launcher\standalone\HMC.exe' then
      begin
        log.Info('Updating Standalone Version');
        if FileExists(MinecraftDir + '\launcher\standalone\HMC.exe.old')
then
          TFile.Delete(MinecraftDir +
'\launcher\standalone\HMC.exe.old');
        TFile.Move(MinecraftDir + '\launcher\standalone\HMC.exe',
MinecraftDir + '\launcher\standalone\HMC.exe.old');
        AddDownload(Settings.FileRepoURL +
'/launcher/standalone/HMC.exe', MinecraftDir +
'\launcher\standalone\HMC.exe', onUpdateDownloaded);
      end;
    if ParamStr(0) = MinecraftDir + '\launcher\user\HMC.exe' then
    begin
      log.Info('Updating User Version');
      if FileExists(MinecraftDir + '\launcher\user\HMC.exe.old') then
        TFile.Delete(MinecraftDir + '\launcher\user\HMC.exe.old');
      TFile.Move(MinecraftDir + '\launcher\user\HMC.exe', MinecraftDir +
'\launcher\user\HMC.exe.old');
      AddDownload(Settings.FileRepoURL + '/launcher/user/HMC.exe',
MinecraftDir + '\launcher\user\HMC.exe', onUpdateDownloaded);
    end;
  Except On E:Exception do
  begin
    Log.Error('Unable to Update Launcher');
    Log.Error(E.ClassName + ':' + E.Message);
  end;
  end;

end;
{$ENDREGION}

end.
```

UnitLogger

```
unit UnitLogger;

//Logging Unit, all others should use this


interface


uses
  System.Classes;


type
  TEntryType = (Info, Warning, Error);


  TOnLogEvent = Procedure (Text : String; EntryType : TEntryType) of
object;
  TLogger = class(TStringList)
  public
    OnLog: TOnLogEvent;
    Procedure Error(Text : String);
    Procedure Info(Text : String);
    Procedure Warn(Text : String);
  end;


implementation


{$REGION 'TLogger'}
//Log an error
Procedure TLogger.Error(Text : String);
begin
  Text := '[Error] ' + Text;
  if Assigned(OnLog) then OnLog(Text, TEntryType.Error);
  Add(Text);
end;


//Log an info message
```

```
Procedure TLogger.Info(Text : String);
begin
  Text := '[Info] '+ Text;
  if Assigned(OnLog) then OnLog(Text, TEntryType.Info);
  Add(Text);
end;

//Log a warning message
Procedure TLogger.Warn(Text : String);
begin
  Text := '[Warning] ' + Text;
  if Assigned(OnLog) then OnLog(Text, TEntryType.Warning);
  Add(Text);
end;

{$ENDREGION}

end.
```

UnitSettings

```
unit UnitSettings;

interface

uses System.Classes, System.Types, System.SysUtils, Windows;

type
  RConfiguration = record
    CustomMinecraftDir: String[100];
    Xms: Integer;
    Xmx: Integer;
    DBUrl: String[100];
    DBUserName: String[20];
```

```
    DBPassword: String[20];
    FileRepoURL: String[100];
    SecondsTillTermination: Integer;
    //for admin mode only
    Alias: String[20];
    CopyTo: String[100];
    ExcludeRegex: String[255];
  end;

  TSettingLoader = Class
  public
    Function GenerateLauncher(Filename : String; Config : RConfiguration)
: Boolean;
    Function LoadFromFile(Filename: String; var Config: RConfiguration):
Boolean;
    Function LoadFromExecutable(Filename: String; var Config:
RConfiguration): Boolean;
    Procedure SaveToFile(Filename: String; Config: RConfiguration);
    Procedure SaveToExecutable(Filename: String; Config: RConfiguration);
  private
    Function RecordToStream(Config: RConfiguration): TMemoryStream;
    Procedure StreamToRecord(MemoryStream : TMemoryStream; var Config :
RConfiguration);
  End;



implementation


uses UnitMain;


{$REGION 'Loading'}


// Load the settings record from the executable file if possible
Function TSettingLoader.LoadFromExecutable(Filename: String;
  var Config: RConfiguration): Boolean;
var
```

```
  FileStream: TFileStream;
  RecordStream: TMemoryStream;
  iSize: Integer;
begin
  Result := False;
  if FileExists(Filename) then
    try
      FileStream := TFileStream.Create(Filename, fmOpenRead);
      RecordStream := TMemoryStream.Create;
      FileStream.Seek(-SizeOf(Integer), soFromEnd);
      FileStream.Read(iSize, SizeOf(iSize));
      if iSize > FileStream.Size then
      begin
        FileStream.Free;
        raise Exception.Create('Settings not found in executable');
      end;
      FileStream.Seek(-iSize, soFromEnd);
      RecordStream.SetSize(iSize - SizeOf(Integer));
      RecordStream.CopyFrom(FileStream, iSize - SizeOf(iSize));
      RecordStream.Seek(0, soFromBeginning);
      StreamToRecord(RecordStream, Config);
      Log.Info('Loaded Settings from ' + Filename);
      Result := True;
      Filestream.Free;
      RecordStream.Free;
    Except On E:Exception do
    begin
      Log.Warn('Failed to Load Settings from ' + Filename);
      Raise;
    end;
    end
  else
    Log.Warn('Failed to Load Settings from ' + Filename);
end;
```

```
//Load record from file
Function TSettingLoader.LoadFromFile(Filename: String;
  var Config: RConfiguration): Boolean;
var
  ConfigFile: File of RConfiguration;
begin
  try
    AssignFile(ConfigFile, Filename);
    Reset(ConfigFile);
    Read(ConfigFile, Config);
    Log.Info('Loaded Settings from ' + Filename);
  Except
    On E: Exception do
      Log.Warn('Unable to read ' + Filename);
  end;
  CloseFile(ConfigFile);
end;
{$ENDREGION}


{$REGION 'Saving'}


//Copies the current launcher, and appends settings to the end
Function TSettingLoader.GenerateLauncher(Filename: string; Config:
RConfiguration) : Boolean;
begin
  Result := False;
  try
    CopyFile(PChar(ParamStr(0)), PChar(Filename), false);
    SettingManager.SaveToExecutable(FileName, Settings);
  Except On E:Exception do Log.Error('Unable to Save Launcher to ' +
Filename);
  end;
end;
```

```
//Save to executable
Procedure TSettingLoader.SaveToExecutable(Filename: String;
  Config: RConfiguration);
var
  FileStream: TFileStream;
  RecordStream: TMemoryStream;
  iSize: Integer;
begin
  if FileExists(Filename) then
    try
      RecordStream := RecordToStream(Config);
      FileStream := TFileStream.Create(Filename, fmOpenWrite or
        fmShareDenyWrite);
      RecordStream.Seek(0, soFromBeginning);
      FileStream.Seek(0, soFromEnd);
      FileStream.CopyFrom(RecordStream, 0);
      iSize := RecordStream.Size + SizeOf(Integer);
      FileStream.Write(iSize, SizeOf(iSize));
      Log.Info('Saved Settings to ' + Filename);
      FileStream.Free;
    Except On E:Exception do Log.Error('Failed to Write Settings to ' +
Filename);
    end
  else
    Log.Error('Failed to Write Settings to ' + Filename);
end;


Procedure TSettingLoader.SaveToFile(Filename: String; Config:
RConfiguration);
var
  ConfigFile: File of RConfiguration;
begin
  try
    AssignFile(ConfigFile, Filename);
    Rewrite(ConfigFile);
```

```
    Write(ConfigFile, Config);
    Log.Info('Saved Settings to ' + Filename);
  Except
    On E: Exception do
      Log.Error('Unable to save ' + Filename);
  end;
  CloseFile(ConfigFile);
end;

{$ENDREGION}

{$REGION 'Record/Stream Conversions}
//Converts a record to a memorystream
Function TSettingLoader.RecordToStream(Config: RConfiguration):
TMemoryStream;
begin
  Result := TMemoryStream.Create;
  Result.Write(Config, SizeOf(RConfiguration));
end;

//Converts the memorystream back to a record
Procedure TSettingLoader.StreamToRecord(MemoryStream : TMemoryStream; var
Config : RConfiguration);
begin
  MemoryStream.Read(Config, SizeOf(RConfiguration));
end;

{$ENDREGION}

end.
```

UnitDatabase

```
unit UnitDatabase;
```

```
interface


uses System.Sysutils, FMX.Types, FireDAC.Comp.Client, FireDAC.Dapt,
FireDAC.Stan.Def, FireDAC.stan.intf, Data.DB,
    FireDAC.Stan.ASync, System.Classes, FMX.grid;


type
  TStandardEvent = Procedure(Sender : TObject) of object;
  TOnRowFetch = Procedure(Sender : TObject; Fields: TFields) of object;
  TOnQuotaChange = Procedure(Sender : TObject; MinutesLeft : Integer) of
object;


  TDBHandler = class(TFDQuery)
  public
    OnNewRequest : TStandardEvent;
    OnNewUser : TStandardEvent;
    OnNewMessage : TStandardEvent;
    OnNewSession : TStandardEvent;
    OnBanned : TStandardEvent;
    OnQuotaUp : TStandardEvent;
    OnQuotaChange : TOnQuotaChange;
    OnRowFetch : TOnRowFetch;
    LocalUserID : Integer;
    MinutesLeft : Integer;
    Constructor Create(AOwner: TComponent);
    Destructor Destroy;
    Procedure InitialiseDB;
    Function DBExists: Boolean;
    Function DBServerExists: Boolean;
    Procedure SetDatabase(DBURL : String; Username: String; Password:
String);
    Function GetUserID(LoginName : String) : Boolean;
    Function GetAlias : String;
    Function isAdmin : Boolean;
    Function isBanned: Boolean;
```

```
    Procedure BeginSession(ComputerName : String = '');
    Procedure EndSession();
    Procedure InsertMessage(MessageText : String);
    Procedure SetAlias(UserID : Integer; Alias : String);
    Procedure RequestAlias(Alias : String);
    Procedure SetAdmin(UserID : Integer);
    Procedure RevokeAdmin(UserID : Integer);
    Procedure SetBanned(UserID : Integer);
    Procedure ProcessRequest(RequestID : Integer);
    Procedure SetUnbanned(UserID : Integer);
    Procedure RequestMod(ModRequest : String);
    Function GetAverageTime(UserID : Integer) : Single;
    Procedure GetBannedPlayers;
    Procedure GetPopularHours;
    Procedure GetTotalPlayers;
    Procedure GetMessages;
    Procedure GetUsers;
    Procedure GetSessions;
    Procedure GetRequests;
    Procedure CreateUser(LoginName : String);
    Procedure EnableBanChecker;
    Procedure EnableQuota;
    Procedure ChangeDefaultQuota(Minutes : Integer);
    Function RequestType(RequestTypeID : Integer) : String;
    Function GetQuota : Integer;
    Procedure SetQuota(UserID : Integer; Quota : Integer);
    Function GetDefaultQuota : Integer;
    Procedure SearchUser(Query : String);
    Procedure SearchSessions(Query : String);
    Procedure SearchMessages(Query : String);
    Procedure SearchRequests(Query : String);
  private
    UpdateTimer : TTimer;
    QuotaTimer : TTimer;
```

```
    SessionID : Integer;
    RequestsLength : Integer;
    Procedure OnDBError(Sender: TObject; const Initiator : IFDStanObject;
var Error : Exception);
    Procedure OnBannedCheckTimer(Sender : TObject);
    Procedure OnQuotaTimer(Sender : TObject);
    Procedure ProcessQuery;
  end;

implementation

uses UnitMain;

{$REGION 'Construction/Destruction/Event Handling'}

//Create MySQL Connection and Query
Constructor TDBHandler.Create(AOwner: TComponent);
begin
  Inherited Create(AOwner);
  Connection := TFDConnection.Create(Self);
  Connection.DriverName := 'MySQL';
  Connection.OnError := OnDBError;
  //Deliberately blocking so that two calls don't overlap and cause a
crash
  ResourceOptions.CmdExecMode := TFDStanAsyncMode.amBlocking;
  OnError := OnDBError;
  UpdateTimer := TTimer.Create(Self);
  UpdateTimer.Interval := 3000;
  QuotaTimer := TTimer.Create(Self);
end;

Destructor TDBHandler.Destroy;
begin
  Inherited Destroy;
  Connection.Destroy;
```

```
  UpdateTimer.Destroy;
  QuotaTimer.Destroy;
end;

//Log Error Messages
Procedure TDBHandler.OnDBError(Sender: TObject; const Initiator :
IFDStanObject; var Error : Exception);
begin
  Log.Error(Error.Message);
end;
{$ENDREGION}

{$REGION 'Database Creation and Validation'}
//If the database exists, delete it. Recreate it. Create the User,
Session, SessionUser, Messages, MessagesUser, RequestTypes, Requests,
RequestsUser tables
Procedure TDBHandler.InitialiseDB;
begin
  if DBServerExists then
  begin
    Connection.ExecSQL('DROP DATABASE IF EXISTS HMC');
    Connection.ExecSQL('CREATE DATABASE HMC');
    Connection.ExecSQL('USE HMC');

    Log.Info('Created HMC Database');

    ExecSQL('CREATE TABLE User(' +
                'UserID INT NOT NULL AUTO_INCREMENT,' +
                'LoginName VARCHAR(20) NOT NULL,' +
                'Alias VARCHAR(20),' +
                'Quota INT DEFAULT 30,' +
                'Admin BOOL DEFAULT 0,' +
                'Banned BOOL DEFAULT 0,' +
                'PRIMARY KEY (UserID))');
    Log.Info('Created Table User');
```

```
    ExecSQL('CREATE TABLE Session(' +
                'SessionID INT NOT NULL AUTO_INCREMENT,' +
                'LogOnTime DATETIME DEFAULT CURRENT_TIMESTAMP,' +
                'LogOffTime DATETIME,' +
                'ComputerName VARCHAR(40),' +
                'PRIMARY KEY (SessionID))');
    Log.Info('CREATE TABLE Session');

    ExecSQL('CREATE TABLE SessionUser(' +
                'SessionID INT NOT NULL,' +
                'UserID INT NOT NULL,' +
                'FOREIGN KEY (SessionID) REFERENCES Session(SessionID),'
+
                'FOREIGN KEY (UserID) REFERENCES User(UserID))');
    Log.Info('Created Table SessionUser');

    ExecSQL('CREATE TABLE Messages(' +
                'MessageID INT NOT NULL AUTO_INCREMENT,' +
                'TimeSent DATETIME DEFAULT CURRENT_TIMESTAMP,' +
                'MessageText VARCHAR(255),' +
                'PRIMARY KEY (MessageID))');
    Log.Info('Created Table Messages');

    ExecSQL('CREATE TABLE MessagesUser(' +
                'MessageID INT NOT NULL,' +
                'UserID INT NOT NULL,' +
                'FOREIGN KEY (MessageID) REFERENCES Messages(MessageID),'
+
                'FOREIGN KEY (UserID) REFERENCES User(UserID))');
    Log.Info('Created Table MessagesUser');

    ExecSQL('CREATE TABLE RequestTypes(' +
                'RequestTypeID INT NOT NULL AUTO_INCREMENT,' +
                'RequestType VARCHAR(20),' +
```

```
                'PRIMARY KEY (RequestTypeID))');
    Log.Info('Created Table RequestTypes');

    ExecSQL('INSERT INTO RequestTypes (RequestType) ' +
            'VALUES ("Mod"), ("Alias")');
    Log.Info('Created Mod and Alias Request Types');

    ExecSQL('CREATE TABLE Requests(' +
                'RequestID INT NOT NULL AUTO_INCREMENT,' +
                'RequestedDate DATETIME DEFAULT CURRENT_TIMESTAMP,' +
                'RequestTypeID INT,' +
                'RequestValue VARCHAR(255),' +
                'RequestProcessed BOOL DEFAULT 0,' +
                'PRIMARY KEY (RequestID),'+
                'FOREIGN KEY (RequestTypeID) REFERENCES
RequestTypes(RequestTypeID))');
    Log.Info('Created Table Requests');

    ExecSQL('CREATE TABLE RequestsUser(' +
                'RequestID INT NOT NULL,' +
                'UserID INT NOT NULL,' +
                'FOREIGN KEY (RequestID) REFERENCES Requests(RequestID),'
+
                'FOREIGN KEY (UserID) REFERENCES User(UserID))');
    Log.Info('Created Table RequestsUser');

    Log.Info('Created all Tables');
  end;
end;

//Checks whether the database exists
Function TDBHandler.DBExists: Boolean;
begin
  try
    Connection.ExecSQL('USE HMC');
```

```
    Result := True;
  Except On E:Exception do
  begin
    Result := False;
  end;
  end;
end;

//Returns true if database is connectable, throws exception and returns
false otherwise
Function TDBHandler.DBServerExists : Boolean;
begin
  if Pos('Server', Connection.Params.Text) > 0 then
    Result := Connection.Ping;
end;

//Set the connection parameters required to execute queries
Procedure TDBHandler.SetDatabase(DBURL: string; Username: String;
Password: String);
begin
  Connection.Params.Clear;
  Connection.Params.DriverID := 'MySQL';
  Connection.Params.Add('Server='+DBURL);
  Connection.Params.UserName := Username;
  Connection.Params.Password := Password;
  Log.Info('Database server set to ' + DBURL);
end;



{$ENDREGION}

{$REGION 'Queries'}
Procedure TDBHandler.SetAdmin(UserID : Integer);
begin
  ExecSQL('UPDATE USER ' +
```

```
          'SET Admin=1 ' +
          'WHERE UserID=' + UserID.toString);
  Log.Info('UserID made Admin: ' + UserID.ToString());
end;

Procedure TDBHandler.RevokeAdmin(UserID : Integer);
begin
  ExecSQL('UPDATE USER ' +
          'SET Admin=0 ' +
          'WHERE UserID=' + UserID.toString);
  Log.Info('UserID made Admin: ' + UserID.ToString());
end;

Procedure TDBHandler.ProcessRequest(RequestID : Integer);
begin
  ExecSQL('UPDATE Requests ' +
          'SET RequestProcessed=1 ' +
          'WHERE RequestID =' + RequestID.toString);
  Log.Info('RequestID Processed: ' + RequestID.ToString());
end;

//SQL insert query to update alias. Throws logger error if User ID has
not been set or created before.
Procedure TDBHandler.SetAlias(UserID : Integer; Alias: String);
begin
  ExecSQL('UPDATE USER ' +
          'SET Alias="' + Alias + '" ' +
          'WHERE UserID=' + UserID.toString);
  Log.Info('Alias updated to ' + Alias);
end;

//Ban the given User ID
Procedure TDBHandler.SetBanned(UserID : Integer);
begin
```

```
  ExecSQL('UPDATE USER ' +
          'SET Banned=1 ' +
          'WHERE UserID=' + UserID.toString);
  Log.Info('User number ' + UserID.toString + ' has been banned');
end;


//Unban the given User ID
Procedure TDBHandler.SetUnbanned(UserID : Integer);
begin
  ExecSQL('UPDATE USER ' +
          'SET Banned=0 ' +
          'WHERE UserID=' + UserID.toString);
  Log.Info('User number ' + UserID.toString + ' has been banned');
end;

//Creates a new database user
Procedure TDBHandler.CreateUser(LoginName: String);
begin
  ExecSQL('INSERT INTO User (LoginName) ' +
          'VALUES ("' + LoginName +'")');
  Log.Info('Created user ' + LoginName);
end;

//Insert new message, and link the generated message id to the userID in
the MessagesUser table
Procedure TDBHandler.InsertMessage(MessageText: String);
begin
  if LocalUserID = 0 then Log.Error('Please ensure that the User ID is
set, by getUserID, or CreateUser')
  else
  begin
  //Escape Quotes
    MessageText := StringReplace(MessageText, '"', '\"', [rfReplaceAll]);
    ExecSQL('INSERT INTO Messages (MessageText) ' +
```

```
            'VALUES ("' + MessageText + '");' +
            'INSERT INTO MessagesUser (MessageID, UserID) ' +
            'VALUES (LAST_INSERT_ID(),' + LocalUserID.toString + ')');
    Log.Info('Message Inserted: ' + MessageText);
  end;
end;

//Adds an alias request, assuming the RequestTypeID is 2
Procedure TDBHandler.RequestAlias(Alias: String);
begin
  if LocalUserID = 0 then Log.Error('Please ensure that the User ID is
set, by getUserID, or CreateUser')
  else
  begin
    ExecSQL('INSERT INTO Requests (RequestTypeID, RequestValue) ' +
            'VALUES (2,"' + Alias + '");' +
            'INSERT INTO RequestsUser (RequestID, UserID) ' +
            'VALUES (LAST_INSERT_ID(),' + LocalUserID.ToString + ')');
    Log.Info('Alias Requested : ' + Alias);
  end;
end;

//Adds a mod request, assuming the RequestTypeID is 1
Procedure TDBHandler.RequestMod(ModRequest: String);
begin
  if LocalUserID = 0 then Log.Error('Please ensure that the User ID is
set, by getUserID, or CreateUser')
  else
  begin
    ExecSQL('INSERT INTO Requests (RequestTypeID, RequestValue) ' +
            'VALUES (1,"' + ModRequest + '");' +
            'INSERT INTO RequestsUser (RequestID, UserID) ' +
            'VALUES (LAST_INSERT_ID(),' + LocalUserID.ToString + ')');
    Log.Info('Mod Request : ' + ModRequest);
  end;
```

```
end;

//Inserts a new session, and then links it to the userid in SessionUsers
Procedure TDBHandler.BeginSession(ComputerName: String);
begin
  ExecSQL('INSERT INTO Session (ComputerName) ' +
          'Values("' + ComputerName + '");' +
          'INSERT INTO SessionUser (SessionID, UserID) ' +
          'VALUES (LAST_INSERT_ID(),' + LocalUserID.ToString + ')');
  Open('Select LAST_INSERT_ID()');
  SessionID := Fields[0].AsInteger;
  Close;
  Log.Info('Started Session ' + SessionID.ToString);
end;

//Ends the previously created session
Procedure TDBHandler.EndSession();
begin
  if SessionID = 0 then Log.Error('There is no session to end')
  else
  begin
    ExecSQL('UPDATE Session ' +
            'SET LogOffTime=NOW() ' +
            'WHERE SessionID=' + SessionID.ToString);
    SessionID := 0;
    Log.Info('Session ended');
  end;
end;

//Returns whether the current user is an admin or not
Function TDBHandler.isAdmin(): Boolean;
begin
  if LocalUserID = 0 then Log.Error('Please ensure that the User ID is
set, by getUserID, or CreateUser')
```

```
  else
  begin
    Open('SELECT Admin ' +
         'FROM User ' +
         'WHERE UserID=' + LocalUserID.ToString());
    Result := Fields[0].AsBoolean;
    Close;
  end;
end;

//Returns Alias
Function TDBHandler.GetAlias(): String;
begin
  if LocalUserID = 0 then Log.Error('Please ensure that the User ID is
set, by getUserID, or CreateUser')
  else
  begin
    Open('SELECT Alias ' +
         'FROM User ' +
         'WHERE UserID=' + LocalUserID.ToString());
    Result := Fields[0].AsString;
    Close;
  end;
end;

//Query that returns the average session time in minutes for given UserID
Function TDBHandler.GetAverageTime(UserID : Integer) : Single;
begin
  Open('SELECT AVG(TIME_TO_SEC(TIMEDIFF(LogOffTime, LogOnTime))/60) ' +
       'FROM Session, SessionUser ' +
       'WHERE SessionUser.SessionID = Session.SessionID ' +
       'AND SessionUser.UserID = ' + UserID.ToString);
  Result := Fields[0].AsSingle;
end;
```

```
//Return Banned
Function TDBHandler.isBanned(): Boolean;
begin
  if LocalUserID = 0 then Log.Error('Please ensure that the User ID is
set, by getUserID, or CreateUser')
  else
  begin
    Open('SELECT Banned ' +
         'FROM User ' +
         'WHERE UserID=' + LocalUserID.ToString());
    Result := Fields[0].AsBoolean;
    Close;
  end;
end;

//Gets a list of banned players
Procedure TDBHandler.GetBannedPlayers;
begin
  Open('SELECT * ' +
       'FROM User ' +
       'WHERE User.Banned = 1');
  ProcessQuery;
  Close;
end;

//Gets Message Sender's name, Message, and the time sent
Procedure TDBHandler.GetMessages;
begin
  Open('SELECT LoginName,Alias,MessageText, TimeSent ' +
       'FROM Messages, MessagesUser, User ' +
       'WHERE Messages.MessageID = MessagesUser.MessageID ' +
       'AND MessagesUser.UserID = User.UserID ' +
       'ORDER BY TimeSent DESC ' +
```

```
        'LIMIT 5000');
  ProcessQuery;
  Close;
end;

//Gets a list of the users, with average time and session count
Procedure TDBHandler.GetUsers;
begin
  Open('SELECT User.UserID, LoginName, Admin, Alias, Banned, Quota,
AVG(TIME_TO_SEC(TIMEDIFF(LogOffTime, LogOnTime))/60) as Average,
COUNT(Session.SessionID) as SessionCount ' +
          'FROM User, SessionUser, Session ' +
        'WHERE Session.SessionID = SessionUser.SessionID ' +
        'AND SessionUser.UserID = User.UserID ' +
        'GROUP BY User.UserID');
  ProcessQuery;
  Close;
end;

//Gets sessions, user etc
Procedure TDBHandler.GetSessions;
begin
  Open('SELECT LoginName, Alias, LogOnTime, LogOffTime, ComputerName ' +
        'FROM Session, SessionUser, User ' +
        'WHERE Session.SessionID = SessionUser.SessionID ' +
        'AND SessionUser.UserID = User.UserID ' +
        'ORDER BY LogOnTime DESC ' +
        'LIMIT 5000');
  ProcessQuery;
  Close;
end;

//Retuns the 5 most popular hours
Procedure TDBHandler.GetPopularHours;
begin
```

```
  Open('SELECT HOUR(LogOnTime) as Hr,COUNT(*) AS Cnt ' +
       'FROM Session ' +
       'GROUP BY Hour(LogOnTime) ' +
       'ORDER BY Cnt DESC ' +
       'LIMIT 5');
  ProcessQuery;
  Close;
end;


Procedure TDBHandler.GetTotalPlayers;
begin
  Open('SELECT COUNT(*) ' +
       'FROM User');
  ProcessQuery;
  Close;
end;

//Stores the UserID as a private class variable
Function TDBHandler.GetUserID(LoginName: String): Boolean;
begin
  Result := False;
  Open('SELECT UserID ' +
       'FROM User ' +
       'WHERE LoginName="' + LoginName + '"');
  if Fields.Count = 0 then Log.Warn('No User ID found for ' + LoginName)
  else
    if Fields[0].AsInteger > 0 then
    begin
      LocalUserID := Fields[0].AsInteger;
      Log.Info('UserID: ' + LocalUserID.ToString());
      Result := True;
    end
  else Result := False;
```

```
  Close;
end;


//Return each row in the form of Fields to onRowFetch and fire event
until no more data
Procedure TDBHandler.ProcessQuery;
begin
  if assigned(onRowFetch) then while not EOF do
    begin
      onRowFetch(Self, Fields);
      Next;
    end;
end;


//Returns the request type of the given ID as a string
Function TDBHandler.RequestType(RequestTypeID: Integer) : String;
begin
  Open('SELECT RequestType ' +
       'FROM RequestTypes ' +
       'WHERE RequestTypeID=' + RequestTypeID.ToString());
  Result := Fields[0].AsString;
  Close;
end;


//Returns name, request and type in one go
Procedure TDBHandler.GetRequests;
begin
  Open('SELECT
User.UserID,LoginName,Requests.RequestID,RequestedDate,RequestValue,Reque
stType ' +
       'FROM User, RequestsUser, Requests, RequestTypes ' +
       'WHERE Requests.RequestID = RequestsUser.RequestID ' +
       'AND RequestsUser.UserID = User.UserID ' +
       'AND RequestProcessed = 0 ' +
       'AND RequestTypes.RequestTypeID = Requests.RequestTypeID');
```

```
  ProcessQuery;
  Close;
end;

//Returns name, request and type in one go
Procedure TDBHandler.SearchRequests(Query : String);
begin
  Open('SELECT
User.UserID,LoginName,Requests.RequestID,RequestedDate,RequestValue,Reque
stType ' +
        'FROM User, RequestsUser, Requests, RequestTypes ' +
        'WHERE Requests.RequestID = RequestsUser.RequestID ' +
        'AND RequestsUser.UserID = User.UserID ' +
        'AND RequestTypes.RequestTypeID = Requests.RequestTypeID ' +
        'AND RequestProcessed = 0 ' +
        'AND (LoginName LIKE "%' + Query + '%" ' +
        'OR RequestedDate LIKE "%' + Query + '%" ' +
        'OR RequestType LIKE "%' + Query + '%" ' +
        'OR RequestValue LIKE "%' + Query + '%") ' +
        'ORDER BY RequestedDate DESC');
  ProcessQuery;
  Close;
end;

//Gets the quota, and resets it if it is a new day
Function TDBHandler.getQuota : Integer;
var
  DateDifference : Integer;
  DefaultQuota : Integer;
begin
  if LocalUserID = 0 then Log.Error('Please ensure that the User ID is
set, by getUserID, or CreateUser')
  else
  begin
    Open('SELECT DateDiff(NOW(), MAX(LogOffTime)) ' +
```

```
            'FROM Session, SessionUser ' +
            'WHERE SessionUser.UserID = ' + LocalUserID.toString + ' ' +
            'AND Session.SessionID = sessionUser.SessionID');
    DateDifference := Fields[0].AsInteger;
    Close;
    Open('SELECT Quota ' +
         'FROM User ' +
         'WHERE UserID=' + LocalUserID.ToString());
    Result := Fields[0].AsInteger;
    Close;
    //If the last login time was not today, and the quota of the user is
less than the default quota, replenish it
    DefaultQuota := GetDefaultQuota;
    if (Result < DefaultQuota) AND (DateDifference > 0) then
    begin
      setQuota(LocalUserID, DefaultQuota);
      Result := DefaultQuota;
    end;
  end;
end;


//Sets the quota of userID to the integer provided
Procedure TDBHandler.SetQuota(UserID : Integer; Quota : Integer);
begin
  ExecSQL('UPDATE User ' +
       'SET Quota=' + Quota.ToString + ' ' +
       'WHERE UserID=' + UserID.ToString);
  Log.Info('Quota for ' + UserID.ToString + ' changed to ' +
Quota.ToString + ' minutes');
end;


//Changes the default quota constraint in SQL, so resetting daily quotas
is done using this value
Procedure TDBHandler.ChangeDefaultQuota(Minutes: Integer);
begin
```

```
  ExecSQL('ALTER TABLE User ' +
          'ALTER COLUMN Quota DROP DEFAULT;' +
          'ALTER TABLE User ' +
          'ALTER COLUMN Quota SET DEFAULT ' + Minutes.ToString);
  Log.Info('Updated Default Quota');
end;

//Grabs default value of quota
Function TDBHandler.GetDefaultQuota : Integer;
begin
  Open('SELECT DEFAULT(Quota) ' +
       'FROM User ' +
       'LIMIT 1');
  Result := Fields[0].AsInteger;
  Close;
end;

//Search all user fields
Procedure TDBHandler.SearchUser(Query: string);
begin
  Open('SELECT User.UserID, LoginName, Admin, Alias, Banned, Quota,
AVG(TIME_TO_SEC(TIMEDIFF(LogOffTime, LogOnTime))/60) as Average,
COUNT(Session.SessionID) as SessionCount ' +
         'FROM User, SessionUser, Session ' +
       'WHERE Session.SessionID = SessionUser.SessionID ' +
       'AND SessionUser.UserID = User.UserID ' +
       'AND (LoginName LIKE "%' + Query + '%" ' +
            'OR Alias LIKE "%' + Query + '%") ' +
       'GROUP BY User.UserID');
  ProcessQuery;
  Close;
end;

//Search all session fields
Procedure TDBHandler.SearchSessions(Query: string);
```

```
begin
  Open('SELECT LoginName, Alias, LogOnTime, LogOffTime, ComputerName ' +
       'FROM Session, SessionUser, User ' +
       'WHERE Session.SessionID = SessionUser.SessionID ' +
       'AND SessionUser.UserID = User.UserID ' +
       'AND (LoginName like "%' + Query + '%" ' +
       'OR LogOnTime like "%' + Query + '%" ' +
       'OR ComputerName like "%' + Query + '%" ' +
       'OR LogOffTime like "%' + Query + '%") ' +
       'ORDER BY LogOnTime DESC');
  ProcessQuery;
  Close;
end;

//Search all message fields
Procedure TDBHandler.SearchMessages(Query: string);
begin
  Open('SELECT LoginName,Alias,MessageText, TimeSent ' +
       'FROM Messages, MessagesUser, User ' +
       'WHERE Messages.MessageID = MessagesUser.MessageID ' +
       'AND MessagesUser.UserID = User.UserID ' +
       'AND (MessageText like "%' + Query + '%" ' +
       'OR TimeSent like "%' + Query + '%" ' +
       'OR LoginName like "%' + Query + '%") ' +
       'ORDER BY TimeSent DESC');
  ProcessQuery;
  Close;
end;
{$ENDREGION}

{$REGION 'Timer Triggers'}
Procedure TDBHandler.OnBannedCheckTimer(Sender: TObject);
begin
  Open('SELECT Banned ' +
```

```
        'FROM User ' +
        'WHERE UserID=' + LocalUserID.ToString);
  if Fields[0].AsBoolean then
  begin
    if assigned(onBanned) then OnBanned(Self);
    Log.Info('Banned');
    UpdateTimer.Enabled := False;
  end;
  Close;
end;

//Checks if Quota is empty, otherwise decrement it
Procedure TDBHandler.onQuotaTimer(Sender : TObject);
begin
  Dec(MinutesLeft);
  if MinutesLeft > 0 then
  begin
    SetQuota(LocalUserID, MinutesLeft);
    if Assigned(OnQuotaChange) then OnQuotaChange(Self, MinutesLeft);
  end
  else
  begin
    if Assigned(onQuotaUp) then OnQuotaUp(Self);
    SetQuota(LocalUserId, 0);
    QuotaTimer.Enabled := False;
  end;
end;

Procedure TDBHandler.EnableBanChecker;
begin
  OnBannedCheckTimer(Self);
  UpdateTimer.OnTimer := OnBannedCheckTimer;
  UpdateTimer.Enabled := True;
end;
```

```
//Checks the quota every minute as long as it is not 0
Procedure TDBHandler.EnableQuota;
begin
  QuotaTimer.OnTimer := OnQuotaTimer;
  QuotaTimer.Interval := 60000;
  MinutesLeft := getQuota;
  if MinutesLeft = 0 then if Assigned(OnQuotaUp) then (OnQuotaUp(Self))
  else
  begin
    QuotaTimer.Enabled := True;
  end;
end;
{$ENDREGION}
end.
```

UnitDownloader

```
unit unitDownloader;

interface
uses
  System.Classes, FMX.Ani, idHTTP, idComponent, System.SysUtils,
FMX.Types,
 idSSLOpenSSL, idTCPClient, XSuperObject, idHashMessageDigest,
system.IOUtils, system.Types;

Function InternetConnected : Boolean;
Function MD5(const FileName : String) : String;

type

TDownloadEvent = Procedure(Sender : TObject;  URL : String) of object;
TDownloadProgressEvent = Procedure(Sender : TObject; URL : String;
Progress : Integer) of object;
TDownloadErrorEvent = Procedure(Sender : TObject; URL : String; Path :
String; E : Exception; OnBegin : TDownloadEvent; OnProgressChange :
```

```
TDownloadProgressEvent; OnDownloadComplete : TDownloadEvent;
OnDownloadCompleteSecondary : TDownloadEvent) of object;

TStandardEvent = Procedure(Sender : TObject) of object;

TNewDownloadEvent = Procedure(Sender : TObject; URL : String; Path :
String; OnDownloadComplete : TDownloadEvent) of object;



//Queue Item record to hold download details

  RQueueItem = Record

    URL : String;

    Path : String;

    OnProgress : TDownloadProgressEvent;

    OnComplete : TDownloadEvent;

    //In case I want to do 2 different things onComplete. Not the best
design though.

    OnCompleteSecondary : TDownloadEvent;

    OnError : TDownloadErrorEvent;

    OnBegin : TDownloadEvent;

  End;


//Basic Queue Structure

  TQueue = Class

    public

      constructor Create;

      Procedure Enqueue(QueueItem : RQueueItem);

      Function Dequeue : RQueueItem;

      Function High : Integer;

      Function Low : Integer;

      Function Length : Integer;

    private

      Queue : array of RQueueItem;

  End;


//Threaded Downloader

  TDownloader = Class(TThread)
```

```
    protected
      HTTP : TIdHTTP;
      CurrentURL : String;
      CurrentPath : String;
      Procedure OnHTTPWork(Sender : TObject; WorkMode : TWorkMode;
WorkCount : Int64);
      Procedure OnFinish(Sender : TObject; WorkMode : TWorkMode);
      Procedure Execute; Override;
      Procedure DownloadFile;
    public
      onComplete : TDownloadEvent;
      onCompleteSecondary : TDownloadEvent;
      onReadyToDownload : TStandardEvent;
      onProgress : TDownloadProgressEvent;
      onError : TDownloadErrorEvent;
      onBegin : TDownloadEvent;
      Property URL : String Read CurrentURL;
      Property Path : String Read CurrentPath;
      Constructor Create;
      Procedure Download(URL : String; Path : String);
  End;



//Download Manager
  TDownloadManager = Class
    public
      OnQueueEmpty : TStandardEvent;
      constructor Create;
      destructor Destroy;
      Procedure AddDownload(URL : String; FilePath : String; OnBegin :
TDownloadEvent = nil; OnProgressChange : TDownloadProgressEvent = nil;
OnDownloadComplete : TDownloadEvent = nil; OnDownloadError :
TDownloadErrorEvent = nil; OnDownloadCompleteSecondary : TDownloadEvent =
nil);
      Function DownloadCount : Integer;
    private
```

```
      Downloaders : array of TDownloader;
      Downloads : TQueue;
      Procedure ProcessQueue;
      Procedure OnDownloadErrorRetry(Sender : TObject; URL : String; Path
: String; E : Exception; OnBegin : TDownloadEvent; OnProgressChange :
TDownloadProgressEvent; OnDownloadComplete : TDownloadEvent;
OnDownloadCompleteSecondary : TDownloadEvent = nil);
      Procedure OnDownloadErrorFail(Sender : TObject; URL : String; Path
: String; E : Exception; OnBegin : TDownloadEvent; OnProgressChange :
TDownloadProgressEvent; OnDownloadComplete : TDownloadEvent;
OnDownloadCompleteSecondary : TDownloadEvent = nil);
      Procedure OnNewDownload(Sender : TObject);
  End;

Const
  DOWNLOADTHREADS = 4;

implementation

uses UnitMain;

{$REGION 'Useful Functions'}

Function MD5(const FileName: string): string; //SO
var
  IdMD5: TIdHashMessageDigest5;
  FS: TFileStream;
begin
 IdMD5 := TIdHashMessageDigest5.Create;
 FS := TFileStream.Create(FileName, fmOpenRead or fmShareDenyWrite);
 try
   Result := IdMD5.HashStreamAsHex(FS)
 finally
   FS.Free;
   IdMD5.Free;
 end;
```

```
end;

Function InternetConnected: Boolean;
var
  Client: TidTCPClient;
begin
  Client := TidTCPClient.Create(nil);
  Client.Host := 'google.com';
  Client.Port := 80;
  try
    Client.Connect;
    result := True;
    Client.Disconnect;
  Except
    On E: Exception do
      result := False
  end;
  Client.Free;
end;

{$ENDREGION}

{$REGION 'TQueue'}
Constructor TQueue.Create;
begin
  inherited Create;
  SetLength(Queue, 0);
end;

//Add a URL to the array
Procedure TQueue.Enqueue(QueueItem : RQueueItem);
begin
  SetLength(Queue, Length + 1);
  Queue[High] := QueueItem;
```

```
end;

//Pop the URL from the array
Function TQueue.Dequeue : RQueueItem;
begin
  Result := Queue[High];
  SetLength(Queue, High);
end;

//Returns the highest index in queue
Function TQueue.High : Integer;
begin
  Result := System.High(Queue);
end;

//Returns lowest index in queue
Function TQueue.Low : Integer;
begin
  Result := System.Low(Queue);
end;

//Returns length of queue
Function TQueue.Length : Integer;
begin
  Result := System.Length(Queue);
end;
{$ENDREGION}

{$REGION 'TDownloader'}
Constructor TDownloader.Create;
begin
  Inherited Create(True);
  HTTP := TIdHTTP.Create(nil);
  HTTP.OnWork := onHTTPWork;
```

```
end;

//Call Download to actually initiate the download
Procedure TDownloader.Download(URL: string; Path: string);
begin
  CurrentURL := URL;
  CurrentPath := Path;
  Start;
end;

//Download given URL to folder. Calls error event if exists.
Procedure TDownloader.DownloadFile;
var
  FileStream : TFileStream;
  SSLHandler : TIdSSLIOHandlerSocketOpenSSL;
begin
  try
    try
      if Assigned(OnBegin) then OnBegin(Self, CurrentURL);
      //Create nested directory if it doesn't exist
      ForceDirectories(ExtractFilePath(CurrentPath));
      FileStream := TFileStream.Create(CurrentPath, fmCreate);
      //if HTTPS then assign SSL handler
      if CurrentURL[5] = 's' then
      begin
        SSLHandler := TIdSSLIOHandlerSocketOpenSSL.Create(HTTP);
        HTTP.IOHandler := SSLHandler;
      end;
      //Put the contents of the URL into the file(stream)
      HTTP.Get(CurrentURL, FileStream);
      //Call onFinish
      OnFinish(Self, TWorkMode.wmWrite);
    Except On E:Exception do
    begin
```

```
//      if Assigned(OnComplete) then OnComplete(Self, CurrentURL);
      if Assigned(OnError) then OnError(Self, CurrentURL, CurrentPath, E,
OnBegin, OnProgress, OnComplete, OnCompleteSecondary);
    end;
    end;
  finally
    FileStream.Free;
    CurrentURL := '';
    CurrentPath := '';
    if Assigned(OnReadyToDownload) then OnReadyToDownload(Self);
  end;
end;


//Calculates percentage and fires event if exists
Procedure TDownloader.OnHTTPWork(Sender: TObject; WorkMode: TWorkMode;
WorkCount: Int64);
var
  ContentLength: Int64;
  Percent: Integer;
begin
  //Kill thread if it has been politely terminated
  if Terminated then Abort;
  ContentLength := HTTP.Response.ContentLength;
  if (Pos('chunked', LowerCase(HTTP.Response.TransferEncoding)) = 0) and
(ContentLength > 0) then
  begin
    Percent := 100 * WorkCount div ContentLength;
    //Update progress bar
    if Assigned(OnProgress) then OnProgress(HTTP, CurrentURL, Percent);
  end;
end;


//Calls onComplete event
Procedure TDownloader.OnFinish(Sender: TObject; WorkMode: TWorkMode);
var
```

```
  URLCopy : String;
begin
  //Create a unique copy of the string in memory, so when the thread
dies, the passed parameters to events do not disappear
  URLCopy := CurrentURL;
  UniqueString(CurrentURL);
  //Used to remove self from GUI. Should block
  if Assigned(OnComplete) then OnComplete(Self, CurrentURL);
  //Asynchronous callback using anonymous Procedure - thread should
continue to terminate without being blocked
  if Assigned(OnCompleteSecondary) then
  Queue(Procedure
        begin
          OnCompleteSecondary(Self, URLCopy)
          end);
  Queue(Procedure
        begin
            Log.Info('Download Complete: ' + URLCopy);
        end);
  CurrentURL := '';
  CurrentPath := '';
end;

//When the thread is started
Procedure TDownloader.Execute;
begin
  DownloadFile;
end;

{$ENDREGION}
     { TODO : Implement auto-Proxy support }

{$REGION 'TDownloadManager'}
Constructor TDownloadManager.Create;
begin
```

```
  inherited Create;
  Downloads := TQueue.Create;
  //Allow for only DOWNLOADTHREAD number of downloaders to be active at a
time. Constant in UnitMain
  SetLength(Downloaders, DOWNLOADTHREADS);
end;

Destructor TDownloadManager.Destroy;
begin
  Setlength(Downloaders, 0);
  Downloads.Destroy;
  Inherited Destroy;
end;

//Returns downloads left
Function TDownloadManager.DownloadCount : Integer;
begin
  Result := Downloads.Length;
end;

//Adds a download to the queue
Procedure TDownloadManager.AddDownload(URL, FilePath: String; OnBegin :
TDownloadEvent; OnProgressChange: TDownloadProgressEvent;
OnDownloadComplete: TDownloadEvent; OnDownloadError :
TDownloadErrorEvent; OnDownloadCompleteSecondary : TDownloadEvent);
var
  QueueItem : RQueueItem;
begin
  QueueItem.URL := URL;
  QueueItem.Path := FilePath;
  QueueItem.OnProgress := OnProgressChange;
  QueueItem.OnComplete := OnDownloadComplete;
  QueueItem.OnCompleteSecondary := OnDownloadCompleteSecondary;
  QueueItem.OnBegin := OnBegin;
  //If there is no on error, then the default behaviour of retry the
download once will occur
```

```
  if not Assigned(OnDownloadError) then QueueItem.OnError :=
OnDownloadErrorRetry else QueueItem.OnError := OnDownloadError;
  Downloads.Enqueue(QueueItem);
  ProcessQueue;
end;

//Checks and assigns downloads to downloaders if they are free
Procedure TDownloadManager.ProcessQueue;
var
  i : Integer;
begin
  //If any of the URLs of the downloaders are blank, give them a new
download
  for i := Low(Downloaders) to High(Downloaders) do
  begin
    if not Assigned(Downloaders[i]) then Downloaders[i] :=
TDownloader.Create;
    if Downloaders[i].URL = '' then
      if Downloads.Length > 0 then
      With Downloads.Dequeue do
      begin
        //Recreate thread. Consider using thread pool later
        FreeAndNil(Downloaders[i]);
        Downloaders[i] := TDownloader.Create;
        Downloaders[i].onProgress := onProgress;
        Downloaders[i].onComplete := onComplete;
        Downloaders[i].onCompleteSecondary := onCompleteSecondary;
        Downloaders[i].onError := OnError;
        Downloaders[i].onReadyToDownload := OnNewDownload;
        Downloaders[i].onBegin := onBegin;
        Downloaders[i].Download(URL, Path);
        Log.Info('Downloading ' + URL + ' to ' + Path);
      end
      else if Assigned(OnQueueEmpty) then OnQueueEmpty(Self);
  end;
```

```
end;


//If the download fails once, give re-add it to the queue

Procedure TDownloadManager.OnDownloadErrorRetry(Sender : TObject; URL :
String; Path : String; E : Exception; OnBegin : TDownloadEvent;
OnProgressChange : TDownloadProgressEvent; OnDownloadComplete :
TDownloadEvent; OnDownloadCompleteSecondary : TDownloadEvent);

begin

  Log.Warn('Failed to download ' + URL + '. Retrying once.');

  (Sender as TDownloader).Queue(Procedure

                                  begin

                                    AddDownload(URL, Path, OnBegin,
OnProgressChange, OnDownloadComplete, OnDownloadErrorFail,
OnDownloadCompleteSecondary);

                                  end);

end;


//If it fails twice, log an error

Procedure TDownloadManager.OnDownloadErrorFail(Sender: TObject; URL:
string; Path: string; E: Exception; OnBegin : TDownloadEvent;
OnProgressChange: TDownloadProgressEvent; OnDownloadComplete:
TDownloadEvent; OnDownloadCompleteSecondary : TDownloadEvent);

begin

  //Logs why the error occurred

  Log.Error('Failed to download ' + URL + ' to ' + Path + ' for the
second time. Aborting.');

  Log.Error(E.ClassName + ' : ' + E.Message);

  (Sender as TDownloader).Queue(ProcessQueue);

end;


//Asks main thread to check for new downloads

Procedure TDownloadManager.OnNewDownload(Sender : TObject);

begin

  //Asynchronous execution

  (Sender as TDownloader).Queue(ProcessQueue);

end;


{$ENDREGION}
```

```
end.
```

UnitWindows

```
unit UnitWindows;


interface


{$IFDEF MSWINDOWS}


uses WinAPI.Windows, System.SysUtils, WinAPI.SHLObj,
System.Types,UnitOverlay,
  System.Classes, FMX.Types, FMX.Dialogs, WinAPI.Messages,
FMX.Platform.WIn, FMX.Graphics, System.Win.Registry;


Function ComputerName: String;
Function RoamingAppDataPath: String;
Function CurrentUsername: String;
Function GetJavaExe : String;


Type
  TKeyEvent = Procedure(Sender : TObject; Key : Char) of object;
  TCloseEvent = Procedure(Sender : TObject) of object;



  TProgram = Class
  private
    TerminateTimer : TTimer;
    HookTimer : TTimer;
    OverlayTimer : TTimer;
    StartInfo: TStartupInfo;
    ProcInfo: TProcessInformation;
    PHandle : THandle;
    FThandle : THandle;
    WHandle : HWND;
```

```
    Procedure CheckTermination(Sender : TObject);
    Procedure CheckKeyPress(Sender : TObject);
    Procedure SetOverlayPosition(Sender : TObject);
  public
    Launched : Boolean;
    OnKeyPress : TKeyEvent;
    OnClose : TCloseEvent;
    Overlay : TfrmOverlay;
    Property ProcessHandle : THandle read PHandle;
    Property ThreadHandle : THandle read FTHandle;
    Function Launch(Command : String) : Boolean;
    Procedure Close;
    Constructor Create;
    Destructor Destroy;
    Procedure EnableRectangle;
  End;


{$ENDIF}

implementation

{$IFDEF MSWINDOWS}
uses UnitMain;

{$REGION 'Useful Functions'}
//Returns Java path for appropriate architecture
Function GetJavaExe : String;
var
  Registry : TRegistry;
begin
  Result := '';
  //Read only access so that HKLM is accessible
  Registry := TRegistry.Create(KEY_READ);
```

```
  Registry.RootKey := HKEY_LOCAL_MACHINE;
  //Access 64bit registry if on 64 bit platform
  if TOSVersion.Architecture = TOSVersion.TArchitecture.arIntelX86 then
Registry.Access := 0
  else Registry.Access := KEY_WOW64_64KEY;
  if Registry.KeyExists('Software\JavaSoft\Java Runtime Environment')
then
    if Registry.OpenKeyReadOnly('Software\JavaSoft\Java Runtime
Environment') then
      if Registry.OpenKeyReadOnly(Registry.ReadString('CurrentVersion'))
        then Result := Registry.ReadString('JavaHome') +
'\bin\javaw.exe';
end;

// Gets current username
Function CurrentUsername: String;
var
  Size : DWORD;
begin
  Size := 1024;
  SetLength(Result, Size);
  if GetUserName(PChar(Result), Size) then
    SetLength(Result, Size - 1)
  else
    RaiseLastOSError;
end;

// Appdata Roaming Path
Function RoamingAppDataPath: string;
const
  SHGFP_TYPE_CURRENT = 0;
var
  path: array [0 .. MaxChar] of char;
begin
  SHGetFolderPath(0, CSIDL_APPDATA, 0, SHGFP_TYPE_CURRENT, @path[0]);
  Result := StrPas(path);
```

```
end;


//Gets computer name
Function ComputerName : String;
var
  buffer: array[0..255] of char;
  size: dword;
begin
  size := Max_ComputerName_Length;
  if GetComputerName(buffer, size) then
    Result := buffer
  else
    Result := ''
end;
{$ENDREGION}


{$REGION 'TLauncher'}
Constructor TProgram.Create;
begin
  Inherited Create;
  //Using Timers for Hooking, so no DLL is required, especially for 32Bit
and 64Bit versions. Does the job. Not great on CPU.
  HookTimer := TTimer.Create(nil);
  HookTimer.Enabled := False;
  HookTimer.OnTimer := CheckKeyPress;
  HookTimer.Interval := 5;
  TerminateTimer := TTimer.Create(nil);
  TerminateTimer.Enabled := False;
  TerminateTimer.OnTimer := CheckTermination;
  TerminateTimer.Interval := 1000;
  //Checks if minecraft is in foreground and hides or shows overlay
depending on this
  OverlayTimer := TTimer.Create(nil);
  OverlayTimer.Enabled := False;
  OverlayTimer.OnTimer := SetOverlayPosition;
```

```
  OverlayTimer.Interval := 500;
end;

Destructor TProgram.Destroy;
begin
  TerminateTimer.Destroy;
  HookTimer.Destroy;
  OverlayTimer.Destroy;
  Close;
  Inherited Destroy;
end;

//Launches a command, sets handles, and begins checking for termination
every second
Function TProgram.Launch(Command: string) : Boolean;
var
  Cmd : String;
  param : integer;
begin
  //Places command into writable memory
  Cmd := Command;
  UniqueString(Cmd);
  //Fill up StartInfo and ProcInfo with known memory size
  FillChar(StartInfo,SizeOf(TStartupInfo),#0);
  FillChar(ProcInfo,SizeOf(TProcessInformation),#0);
  StartInfo.cb := SizeOf(TStartupInfo);
  //True if the process was executed - See Windows API documentation
  Result := CreateProcess(nil, PChar(Cmd), nil, nil,False,
              CREATE_NEW_PROCESS_GROUP or NORMAL_PRIORITY_CLASS,
              nil, nil, StartInfo, ProcInfo);
  FTHandle := ProcInfo.hThread;
  PHandle := ProcInfo.hProcess;
  //Wait for program to be ready
  WaitForInputIdle(ProcInfo.hProcess, INFINITE);
```

```
  //Grab Window Handle
  Param := ProcInfo.dwThreadId;
//  EnumWindows(CheckWindowHandle, Param);
  //Begin checking for termination
  TerminateTimer.Enabled := True;
  //Begin keylogging
  HookTimer.Enabled := True;
  Launched := True;
end;

//Try to cleanly shutdown program, else terminate it
Procedure TProgram.Close;
var
  ExitCode : DWORD;
begin
  if Launched then
  begin
    //Post a friendly shutdown message and give it 2 seconds to respond
    PostThreadMessage(ProcInfo.dwThreadId, WM_CLOSE, 0, 0);
    WaitForSingleObject(ProcInfo.hProcess, 2000);
    // Check exit code
    ExitCode := 0;
    GetExitCodeProcess(ProcInfo.hProcess, ExitCode);
    if ExitCode = STILL_ACTIVE then TerminateProcess(ProcInfo.hProcess,
0);
  end;
  //The termination checker timer will fire the onClose event
end;

//Checks for a keypress from the launcher window, and fires keypress
event if this is true
Procedure TProgram.CheckKeyPress(Sender: TObject);
var
  ScanCode : Integer;
  KeyResult : array [0..1] of Char;
```

```
  UnicodeResult : Integer;
  KeyboardState : TKeyboardState;
  Key : Integer;
  ThreadID, ProcessID : HWND;
  ShiftPressed : Boolean;
begin
  ThreadID := GetWindowThreadProcessID(GetForegroundWindow, @ProcessID);
  //If the current window is the one created earlier
  if ProcessID = ProcInfo.dwProcessId then
  begin
    //Update ThreadID with real ThreadID of Java window
    procInfo.dwThreadId := ThreadID;
    //Check if any of the keys are pressed. Goes through all keys except
mouse keys
    for Key := VK_BACK to VK_OEM_CLEAR do if GetAsyncKeyState(Key) = -
32767 then
    begin
      if GetAsyncKeyState(VK_SHIFT) = -32768 then ShiftPressed := True
else ShiftPressed := False;
      //Get a Scan code
      ScanCode := MapVirtualKeyExW(Key, MAPVK_VK_TO_VSC_EX,
GetKeyboardLayout(ThreadID));
      if ScanCode > 0 then
      begin
        GetKeyboardState(KeyboardState);
        //Manually modify our simulated keyboard state to include shifts.
This is because GetKeyboardState doesn't work properly
        //If Shift is pressed
        if ShiftPressed then
          //Set the upper bits high
          KeyboardState[VK_Shift] := $80;
          KeyboardState[VK_LShift] := $80;
        end;
      //Do the conversion to a character
      UnicodeResult := ToUnicodeEx(Key, ScanCode, KeyboardState,
KeyResult, SizeOf(KeyResult), 0, GetKeyboardLayout(ThreadID));
```

```
      if UnicodeResult > 0 then if Assigned(OnKeyPress) then
OnKeyPress(Self, KeyResult[0])
      else UnicodeResult := ToUnicodeEx(Key, ScanCode, KeyboardState,
KeyResult, SizeOf(KeyResult), 0, GetKeyboardLayout(ThreadID));
    end;
  end;
end;

//Checks whether the process has terminated and fires event if it has
Procedure TProgram.CheckTermination(Sender: TObject);
begin
  //If the process has been terminated somehow
  if waitForSingleObject(ProcInfo.hProcess, 0) = Wait_Object_0 then
  begin
    CloseHandle(ProcInfo.hProcess);
    CloseHandle(ProcInfo.hThread);
    HookTimer.Enabled := False;
    TerminateTimer.Enabled := False;
    if Assigned(OnClose) then OnClose(Self);
  end;
end;

Procedure TProgram.SetOverlayPosition(Sender: TObject);
var
  Window : TRect;
begin
  //If the current window is the one created earlier
  if GetWindowThreadProcessID(GetForegroundWindow) = ProcInfo.dwThreadId
then
  begin
    //Update window handle
    WHandle := GetForegroundWindow;
    //Get current positon of program
    GetWindowRect(WHandle, Window);
    Overlay.Show;
```

```
    //Set the overlay a tiny bit lower
    Overlay.Width := 350;
    Overlay.Height := 50;
    Overlay.Top := Window.Top + 10;
    Overlay.Left := Window.Left + (Window.Width - Overlay.Width) div 2 ;
      end
  else Overlay.Hide;
end;


//Enables the rectangle overlay
Procedure TProgram.EnableRectangle;
begin
  OverlayTimer.Enabled := true;
  //Force window to be above all others and resize it to fit in centre
top
  SetWindowPos(FMXHANDLETOHWND(overlay.Handle),HWND_TOPMOST, 0 , 0,
Overlay.Width, Overlay.Height,SWP_NOACTIVATE or SWP_NOMOVE or
SWP_NOSIZE);
end;
{$ENDREGION}


{$ENDIF}


end.
```

UnitLauncher

```
unit UnitLauncher;


interface


uses   UnitWindows, UnitOverlay, SuperObject,
      System.SysUtils, System.Classes, System.Types, FMX.Graphics,
FMX.Dialogs, System.UITypes, FMX.Types;


type
//Class that generates command, given Minecraft Directory, Xmx, Xms
```

```
TCommandGenerator = Class
  public
    Xms : Integer;
    Xmx : Integer;
    MinecraftDir : String;
    UserName : String;
    Function GenerateCommand : String;
  private
    MCVersion : String;
    VersionID : String;
    Command : String;
    LauncherProfiles : ISuperObject;
    VersionJSON : ISuperObject;
    Procedure ReplaceArguments;
    Procedure ReplaceArch;
    Procedure ReplaceUsername;
    Procedure ReplaceVersionName;
    Procedure ReplaceGameDirectory;
    Procedure ReplaceAssetsRoot;
    Procedure ReplaceGameAssets;
    Procedure ReplaceAssetIndexName;
    Procedure ReplaceAuthUUID;
    Procedure ReplaceAccessToken;
    Procedure ReplaceUserProperties;
    Procedure ReplaceUserType;
    Function Libraries(JSON : ISuperObject) : String;
    Function MainClass : String;
    Function MinecraftArguments : String;
    Function MinecraftJar : String;
End;

TKeyEvent = Procedure(Sender : TObject; Key : Char) of object;
TCloseEvent = Procedure(Sender : TObject) of object;
```

```
  TSendMessageEvent = Procedure(Sender : TObject; Text : String) of
object;


//Class that monitors and launches the Minecraft instance. Includes
keylogger and drawing dialogs.
  TMinecraftInstance = Class
    public
      OnClose : TCloseEvent;
      OnSendMessage : TSendMessageEvent;
      Constructor Create;
      Destructor Destroy;
      Procedure Launch(Username : String; Xms : Integer; Xmx : Integer;
MinecraftDir : String);
      Procedure SetMessageBox(Text : String);
      Procedure SetLavaOverlay;
      Procedure TerminateIn(Seconds : Integer);
      Procedure Close;
    private
      Generator : TCommandGenerator;
      CountdownTimer : TTimer;
      SecondsLeft : Integer;
      Minecraft : TProgram;
      CurrentMessage : String;
      TypingMessage : Boolean;
      Overlay : TfrmOverlay;
      Procedure ChainOnClose(Sender : TObject);
      Procedure RecordMessage(Sender : TObject; Key : Char);
      Procedure CountDown(Sender : TObject);
  End;



implementation


uses UnitMain;
```

```
{$REGION 'TCommandGenerator'}
Function TCommandGenerator.GenerateCommand: String;
var
  Profile : ISuperObject;
begin
  //Gets version from Launcher_profiles.json
  LauncherProfiles := TSuperObject.ParseFile(MinecraftDir +
'\launcher_profiles.json', False);
  MCVersion := LauncherProfiles.S['MCVersion'];
  //Version ID is the same as Minecraft Version unless Forge is installed
  VersionID := MCVersion;
  if LauncherProfiles.AsObject.Exists('profiles') then
    for Profile in LauncherProfiles['profiles'] do
    begin
    //Forge support
      if Pos('Forge', Profile.AsString) > 0 then
        if Pos(MCVersion, Profile.S['lastVersionId']) > 0 then
          VersionID := Profile.S['lastVersionId'];
      //Liteloader support
      if Pos('LiteLoader', Profile.AsString) > 0 then
        if Pos(MCVersion, Profile.AsString) > 0 then
        begin
          VersionID := Profile.S['lastVersionId'];
          Break;
        end;
    end;
  //JSON containing how to launch Minecraft
  VersionJSON := TSuperObject.ParseFile(MinecraftDir + '\versions\' +
VersionId + '\' + VersionId + '.json', False);


  Command := GetJavaExe + ' -Xms' + Xms.ToString + 'M -Xmx' +
Xmx.ToString + 'M -Dfml.ignoreInvalidMinecraftCertificates=true -
Dfml.ignorePatchDiscrepancies=true ' +
            '-Djava.library.path=' + MinecraftDir + '\versions\' +
MCVersion + '\natives -cp ' + Libraries(VersionJSON) +
```

```
            ';' + MinecraftJar + ' ' + MainClass + ' ' +
MinecraftArguments;
  ReplaceArguments;
  Log.Info(Command);
  Result := Command;
end;

{$REGION 'Argument Replacement'}
{ Add new arguments here }
//Replaces all the arguments in the form of ${argument}
Procedure TCommandGenerator.ReplaceArguments;
begin
  ReplaceArch;
  ReplaceAssetIndexName;
  ReplaceAssetsRoot;
  ReplaceAuthUUID;
  ReplaceGameDirectory;
  ReplaceUserName;
  ReplaceUserProperties;
  ReplaceUserType;
  ReplaceVersionName;
  ReplaceAccessToken;
  ReplaceArch;
end;

//Replaces ${assets_index_name} with the version of Minecraft
Procedure TCommandGenerator.ReplaceAssetIndexName;
begin
  Command := StringReplace(Command, '${assets_index_name}', MCVersion,
[rfReplaceAll]);
end;

//Replace ${game_assets} with legacy path
Procedure TCommandGenerator.ReplaceGameAssets;
begin
```

```
  Command := StringReplace(Command, '${game_assets}', MinecraftDir +
'\assets\virtual\legacy', [rfReplaceAll]);
end;


//Replace ${assets_root} with new assets path
Procedure TCommandGenerator.ReplaceAssetsRoot;
begin
  Command := StringReplace(Command, '${assets_root}', MinecraftDir +
'\assets', [rfReplaceAll]);
end;


//Replace ${auth_uuid} with (Default)
Procedure TCommandGenerator.ReplaceAuthUUID;
begin
  Command := StringReplace(Command, '${auth_uuid}', '(Default)',
[rfReplaceAll]);
end;


//Replace ${game_directory} with MinecraftDir
Procedure TCommandGenerator.ReplaceGameDirectory;
begin
  Command := StringReplace(Command, '${game_directory}', MinecraftDir,
[rfReplaceAll]);
end;


//Replace ${auth_player_name} with Username
Procedure TCommandGenerator.ReplaceUsername;
begin
  Command := StringReplace(Command, '${auth_player_name}', Username,
[rfReplaceAll]);
end;


//Replace ${user_properties} with {}
Procedure TCommandGenerator.ReplaceUserProperties;
begin
```

```
  Command := StringReplace(Command, '${user_properties}', '{}',
[rfReplaceAll]);
end;


//Replace ${user_type} with mojang
Procedure TCommandGenerator.ReplaceUserType;
begin
  Command := StringReplace(Command, '${user_type}', 'mojang',
[rfReplaceAll]);
end;


//Replace ${version_name} with MCVersion
Procedure TCommandGenerator.ReplaceVersionName;
begin
  Command := StringReplace(Command, '${version_name}', MCVersion,
[rfReplaceAll]);
end;


//Replace ${auth_access_token} with FML or anything
Procedure TCommandGenerator.ReplaceAccessToken;
begin
  Command := StringReplace(Command, '${auth_access_token}', 'FML',
[rfReplaceAll]);
end;



//Replaces arch with architecture. Only really applicable to Windows
Procedure TCommandGenerator.ReplaceArch;
var
  Arch : string;
begin
  if TOSVersion.Architecture = TOSVersion.TArchitecture.arIntelX86 then
Arch := '32' else Arch := '64';
  Command := StringReplace(Command, '${arch}', arch, [rfReplaceAll]);
end;
```

```
{$ENDREGION}


//Add Minecraft Jar
Function TCommandGenerator.MinecraftJar : String;
begin
  Result := MinecraftDir + '\versions\' + MCVersion + '\' + MCVersion +
'.jar';
end;


//Adds Unparsed Minecraft Arguments
Function TCommandGenerator.MinecraftArguments;
begin
  Result := VersionJSON.S['minecraftArguments'];
end;


//Gets MainClass
Function TCommandGenerator.MainClass : String;
begin
  Result := VersionJSON.S['mainClass'];
end;


//Returns a list of libraries. Recursive if inheritsFrom is present
Function TCommandGenerator.Libraries(JSON : ISuperObject) : String;
var
  Item, RuleItem : ISuperObject;
  InheritsFromVersion, Version, Package, Name, Path : String;
  SplitPosition : Integer;
begin
  //If details are inherited from another file, load all the libraries
from it by recursive call
  if JSON.asObject.Exists('inheritsFrom') then
  begin
    InheritsFromVersion := JSON.S['inheritsFrom'];
    Result := Result + Libraries(TSuperObject.ParseFile(MinecraftDir +
'\versions\' + InheritsFromVersion + '\' + InheritsFromVersion + '.json',
False));
```

```
  end;


  //Traverse libraries and do not include if it is a native
  for Item in JSON['libraries'] do
  begin
  //Put everything in version
    Version := Item.S['name'];
    SplitPosition := Pos(':', Version);
    //Get Package name
    Package := Copy(Version, 1, SplitPosition - 1);
    //Split Version again
    Delete(Version, 1, SplitPosition);
    //Find the next colon and make it the name of the package
    SplitPosition := Pos(':', Version);
    Name := Copy(Version, 1, SplitPosition - 1);
    Delete(Version, 1, SplitPosition);
    //Replace . with / in package
    Package :=  StringReplace(Package, '.', '\', [rfReplaceAll]);
    //Generate path - convert . to \
    Path := MinecraftDir + '\libraries\' + Package + '\' + Name + '\' +
Version + '\' + Name + '-' + Version + '.jar';
    //Add semicolon at beginning or not depending on whether there is
anything else in string
    if Length(Result) > 0 then Path := ';' + Path;
    if (not Item.AsObject.Exists('natives')) then
    begin
    //Check if it meets rules
     if Item.AsObject.Exists('rules')then
      for RuleItem in Item['rules'] do
        begin
    {$IFDEF MSWINDOWS}
      //Add correct libraries for windows
        if RuleItem.S['action'] = 'disallow' then
          if RuleItem.AsObject.Exists('os') then
            if not(RuleItem.O['os'].S['name'] = 'windows') then
```

```
            begin
              Result := Result + Path;
              Break;
            end;

        if RuleItem.S['action'] = 'allow' then
          if RuleItem.AsObject.Exists('os') then
            if RuleItem.O['os'].S['name'] = 'windows' then
            begin
              Result :=  Result + Path;
              Break;
            end
            else //drop out of this block
          else
          begin
            Result := Result + Path;
            Break;
          end;
    {$ENDIF}
    {$IFDEF MACOS}
      //If I need to extend for MAC OS , insert here
    {$ENDIF}
      //Add it anyway if its not on the rules
      end else
      //ClientReq must be true, or serverreq and clientreq should not
exist
      if (item.AsObject.Exists('serverreq')) or
(item.AsObject.Exists('clientreq')) then
        if (Item.B['clientreq']) or (Item.B['serverreq']) then Result :=
Result + Path
        else //escape else
      else Result := Result + Path;
    end;
  end;
end;
```

```
{$ENDREGION}


{$REGION 'TMinecraftInstance'}


Constructor TMinecraftInstance.Create;
begin
  inherited Create;
  //Create application handling class
  Minecraft := TProgram.Create;
  Minecraft.OnClose := ChainOnClose;
  Minecraft.OnKeyPress := RecordMessage;
  //Overlay is child of main form. Refractor so that this is accessed via
create
  Overlay := TfrmOverlay.Create(frmMain);
  CountdownTimer := TTimer.Create(nil);
  CountdownTimer.Interval := 1000;
  CountdownTimer.Enabled := False;
  CountdownTimer.OnTimer := CountDown;
  //Create command generator
  Generator := TCommandGenerator.Create;
end;


Destructor TMinecraftInstance.Destroy;
begin
  Overlay.Destroy;
  Minecraft.Destroy;
  CountdownTimer.Destroy;
  Generator.Destroy;
  inherited Destroy;
end;


//Chained Procedure to close minecraft
Procedure TMinecraftInstance.Close;
begin
```

```
  CountdownTimer.Enabled := False;
  Minecraft.Close;
end;

//Launch Minecraft
Procedure TMinecraftInstance.Launch(Username : String; Xms : Integer; Xmx
: Integer; MinecraftDir : String);
begin
  Generator.Xms := Xms;
  Generator.Xmx := Xmx;
  Generator.MinecraftDir := MinecraftDir;
  Generator.UserName := UserName;
  Minecraft.Launch(Generator.GenerateCommand);
end;

//Pass on the onclose event - hacky
Procedure TMinecraftInstance.ChainOnClose(Sender: TObject);
begin
  CountdownTimer.Enabled := False;
  if Assigned(OnClose) then OnClose(Self);
end;

//Record the current message in Minecraft. Works by detecting t or /, and
waiting till enter or ESC
Procedure TMinecraftInstance.RecordMessage(Sender: TObject; Key: Char);
begin
  if TypingMessage then
    case Key of
      //Enter key
      chr(13):
      begin
        if CurrentMessage.Length > 0 then if Assigned(OnSendMessage) then
OnSendMessage(Self, CurrentMessage);
        TypingMessage := False;
        CurrentMessage := '';
```

```
      end;
      //Backspace key - Delete 1 character
      chr(8): CurrentMessage := Copy(CurrentMessage, 1 ,
CurrentMessage.Length - 1);
      //Escape key - Destroy Message
      chr(27):
      begin
        TypingMessage := False;
        CurrentMessage := '';
      end;
      else CurrentMessage := CurrentMessage + Key;
    end
    else if (Key = 't') or (Key='/') then
    begin
    //Start a new message
      CurrentMessage := '';
      TypingMessage := True;
    end;
end;

//Sets the text for the overlay
Procedure TMinecraftInstance.SetMessageBox(Text: string);
begin
  //Pass on form to OS specific units, and draw overlay appropriately
  Minecraft.Overlay := Overlay;
  Minecraft.EnableRectangle;
  Overlay.SetText(Text);
end;

//Sets the lava overlay image
Procedure TMinecraftInstance.setLavaOverlay;
begin
  Overlay.ChangeToLava;
end;
```

```
//Begins a termination countdown and then closes Minecraft
Procedure TMinecraftInstance.TerminateIn(Seconds: Integer);
begin
  if Seconds > 0 then
  begin
    SecondsLeft := Seconds;
    CountdownTimer.Enabled := True;
    Overlay.SetText('Closing in ' + SecondsLeft.toString);
  end else Minecraft.Close;
end;

//Decrements timer until 0, then closes Minecraft
Procedure TMinecraftInstance.CountDown(Sender: TObject);
begin
  if SecondsLeft > 0 then
  begin
    Dec(SecondsLeft);
    Overlay.SetText('Closing in ' + SecondsLeft.toString);
  end else Minecraft.Close;
end;

{$ENDREGION}


end.
```

UnitOverlay

```
unit UnitOverlay;

interface

uses
```

```
  System.SysUtils, System.Types, System.UITypes, System.Classes,
System.Variants,
  FMX.Types, FMX.Controls, FMX.Forms, FMX.Graphics, FMX.Dialogs,
FMX.Objects,
  FMX.Ani;

type
  TfrmOverlay = class(TForm)
    BitmapAnimation: TBitmapAnimation;
    txtRemaining: TText;
  public
    Procedure ChangeToLava;
    Procedure SetText(Text : String);
  end;

implementation

{$R *.fmx}

//Enable Lava animation
Procedure TfrmOverlay.ChangeToLava;
begin
  BitmapAnimation.Enabled := True;
end;

//Setter
Procedure TfrmOverlay.SetText(Text : String);
begin
  txtRemaining.Text := Text;
end;

end.
```

UnitUpdater

```
unit UnitUpdater;


interface


uses UnitDownloader,

  System.SysUtils, System.Classes, System.Types, FMX.Types, SuperObject,
System.Zip, System.IOUtils;


Type

TStringListEvent = Procedure(Sender : TObject; List : TStringList) of
object;


  TMinecraftUpdater = class

    public

      OnNewDownload : TNewDownloadEvent;

      OnGetVersions : TStringListEvent;

      MinecraftDir : String;

      Constructor Create;

      Destructor Destroy;

      Procedure GetVersions;

      Procedure Install(Version : String);

    private

      DVersion : String;

      Procedure OnDownloadVersionsJSON(Sender : TObject; URL : String);

      Procedure OnDownloadMinecraftJar(Sender : TObject; URL : String);

      Procedure OnDownloadVersionInformation(Sender : TObject; URL :
String);

      Procedure OnDownloadAssetInformation(Sender : TObject; URL :
String);

      Procedure OnCompressedFile(Sender : TObject; URL : String);

      Procedure CheckNatives(var URL : String; var Path : String);

  end;


Const
```

```
  VERSIONSURL =
'http://s3.amazonaws.com/Minecraft.Download/versions/versions.json';
  LIBRARYURL = 'https://libraries.minecraft.net';
  ASSETURL = 'http://resources.download.minecraft.net';

implementation

uses UnitMain;

{$REGION 'Useful Functions'}
//Gets pos from the right. Credits Vertical Software 1999
Function PosRight(const SubString, SearchString: String): Integer;
var
 i, SearchStringLength, SubstringLength: Integer;
 FirstChar: String[1];
begin
 Result := 0;
 FirstChar := Substring[1];
 SubStringLength := Length(SubString);
 SearchStringLength := Length(SearchString);
 for i := SearchStringLength downto 1 do
   begin
     if SearchString[i] = FirstChar then
       begin
         // check whether there is enough space for the substring to
match
         if SearchStringLength - i < SubStringLength then
           continue;
         if Copy(SearchString, i, SubStringLength) = SubString then
           begin
             Result := i;
             Exit;
           end;
       end;
   end;
```

```
end;


{$ENDREGION}


Constructor TMinecraftUpdater.Create;
begin
  Inherited Create;
end;


Destructor TMinecraftUpdater.Destroy;
begin
  Inherited Destroy;
end;


{$REGION 'Version Processing'}
//Downloads list of minecraft versions, and fires OnGetVersions,
containing a stringlist of the versions
Procedure TMinecraftUpdater.GetVersions;
begin
  if Assigned(OnNewDownload) then
  begin
    if MinecraftDir.Length > 0 then
    begin
      OnNewDownload(Self, VERSIONSURL, MinecraftDir +
'\versions\versions.json', OnDownloadVersionsJSON);
    end else Log.error('No Minecraft Directory Provided');
  end
  else Log.Error('Downloader must be attached');
end;


//Parses JSON file and returns a list of Minecraft full releases
Procedure TMinecraftUpdater.OnDownloadVersionsJSON(Sender : TObject; URL
: String);
var
  List : TStringList;
```

```
  JSON, Version : ISuperObject;
begin
  Log.Info('Parsing versions.json');
  List := TStringList.Create;
  JSON := TSuperObject.ParseFile(MinecraftDir +
'\versions\versions.json', False);
  //for every version, add it to the list if its a release version that
is or above 1.6.4
  for Version in JSON['versions'] do
    if (Version.S['type'] = 'release') and (Version.S['id'] >= '1.6.4')
then List.Add(Version.S['id']);
  //Submit compiled list
  if Assigned(OnGetVersions) then OnGetVersions(Self, List);
  List.Free;
end;


{$ENDREGION}


{$REGION 'Apply Update'}
//Downloads Minecraft version specified
Procedure TMinecraftUpdater.Install(Version: string);
var
  VersionURL : String;
begin
  if Assigned(OnNewDownload) then
  begin
    if MinecraftDir.Length > 0 then
    begin
      DVersion := Version;
      //Build URL string
      VersionURL :=
'http://s3.amazonaws.com/Minecraft.Download/versions/' + Version + '/' +
Version + '.jar';
      //Download Minecraft Jar file from
http://s3.amazonaws.com/Minecraft.Download/versions/<version>/<version>.j
ar to <version>/<version>.jar
```

```
      OnNewDownload(Self, VersionURL, MinecraftDir + '\versions\' +
Version + '\' + Version + '.jar', OnDownloadMinecraftJar);

      //Download information about how to run it

      VersionURL :=
'http://s3.amazonaws.com/Minecraft.Download/versions/' + Version + '/' +
Version + '.json';

      OnNewDownload(Self, VersionURL, MinecraftDir + '\versions\' +
Version + '\' + Version + '.json', OnDownloadVersionInformation);

    end else Log.error('No Minecraft Directory Provided');

  end

  else Log.Error('Downloader must be attached');

end;


//Updates Launcher_profiles.json for forge compatability

Procedure TMinecraftUpdater.OnDownloadMinecraftJar(Sender: TObject; URL:
string);

var

  ProfilesFileName : String;

  JSON : ISuperObject;

begin

  ProfilesFileName := MinecraftDir + '\launcher_profiles.json';

  //Create a new JSON file

  JSON := SO('{}');

  //Add profiles section

  if not JSON.AsObject.Exists('profiles') then JSON.O['profiles'] :=
SO('{}');

  //Store Minecraft Version in launcher_profiles for convenience

  JSON.S['MCVersion'] := DVersion;

  JSON.SaveTo(ProfilesFileName, True, True);

end;



//Downloads the relevant libraries

Procedure TMinecraftUpdater.OnDownloadVersionInformation(Sender: TObject;

  URL: String);

var
```

```
  JSON, Item, RuleItem : ISuperObject;
  Package, Name, Version, Arch, FileName, DownloadURL, Path : String;
  SplitPosition : Integer;
  OnNeedsExtracting : TDownloadEvent;
  Download : Boolean;
begin
  Download := False;
  //Open MinecraftDir\versions\<version>\<version>.json
  JSON := TSuperObject.ParseFile(MinecraftDir + '\versions\' + DVersion +
'\' + DVersion + '.json', false);
  //Download assets
  if JSON.AsObject.Exists('assets') then
    OnNewDownload(Self,
'https://s3.amazonaws.com/Minecraft.Download/indexes/' + JSON.S['assets']
+ '.json', MinecraftDir + '\assets\indexes\' + JSON.S['assets'] +
'.json', OnDownloadAssetInformation)
  else
    OnNewDownload(Self,
'https://s3.amazonaws.com/Minecraft.Download/indexes/legacy.json',
MinecraftDir + '\assets\indexes\legacy.json',
OnDownloadAssetInformation);


  if TOSVersion.Architecture = TOSVersion.TArchitecture.arIntelX86 then
Arch := '32' else Arch := '64';
  //Downloads Libraries
  for Item in JSON['libraries'] do
  begin
  //Put everything in version
    Version := Item.S['name'];
    SplitPosition := Pos(':', Version);
    //Get Package name
    Package := Copy(Version, 1, SplitPosition - 1);
    //Split Version again
    Delete(Version, 1, SplitPosition);
    //Find the next colon and make it the name of the package
    SplitPosition := Pos(':', Version);
    Name := Copy(Version, 1, SplitPosition - 1);
```

```
    Delete(Version, 1, SplitPosition);

    //Replace . with / in package

    Package :=  StringReplace(Package, '.', '\', [rfReplaceAll]);

    //Generate
https://libraries.minecraft.net/<package>/<name>/<version>/<name>-
<version>.jar

    DownloadURL := LIBRARYURL + '/' + Package + '/' + Name + '/' +
Version + '/' + Name + '-' + Version + '.jar';

    DownloadURL := StringReplace(DownloadURL, '\', '/', [rfReplaceAll]);

    //Generate path - convert . to \

    Path := MinecraftDir + '\libraries\' + Package + '\' + Name + '\' +
Version + '\' + Name + '-' + Version + '.jar';

    OnNeedsExtracting := nil;

    //Find out what to download by looking at the rules

    if Item.AsObject.Exists('rules') then

      for RuleItem in Item['rules'] do

      begin

    {$IFDEF MSWINDOWS}

      //Download correct libraries for windows

      //Check if library is on allow list, or if nothing is on the allow
list, or if its not on the disallow list. Restrcture in future.

        if RuleItem.S['action'] = 'disallow' then

          if RuleItem.AsObject.Exists('os') then

            if not(RuleItem.O['os'].S['name'] = 'windows') then

            begin

              Download := True;

              Break;

            end;


        if RuleItem.S['action'] = 'allow' then

          if RuleItem.AsObject.Exists('os') then

            if RuleItem.O['os'].S['name'] = 'windows' then

            begin

              Download := True;

              Break;

            end
```

```
            else //drop out of this block
          else
          begin
            Download := True;
            Break;
          end;
    {$ENDIF}
    {$IFDEF MACOS}
      //If I need to extend for MAC OS , insert here
    {$ENDIF}
      end else Download := True;
      //Otherwise if there are no rules download it anyway
    //If there is a natives object, then tweak URL so it is downloadable,
and save in versions/<version>/natives
    if Download then
      if Item.AsObject.Exists('natives') then
      begin
        //Generate
https://libraries.minecraft.net/<package>/<name>/<version>/<name>-
<version>.jar
        DownloadURL := LIBRARYURL + '/' + Package + '/' + Name + '/' +
Version + '/' + Name + '-' + Version + '-' + Item['natives'].S['windows']
+ '.jar';
        DownloadURL := StringReplace(DownloadURL, '\', '/',
[rfReplaceAll]);
        //Generate path - convert . to \
        Path := MinecraftDir + '\versions\' + DVersion + '\natives\' +
Name + '-' + Version + '-' + Item['natives'].S['windows'] + '.jar';
        //Replace all $arch variables if there are any with corresponding
architecture
        DownloadURL := StringReplace(DownloadURL, '${arch}', Arch,
[rfReplaceAll]);
        Path := StringReplace(Path, '${arch}', Arch, [rfReplaceAll]);
        //Ensure updater extracts the file, as it is really a zip file
        OnNeedsExtracting := OnCompressedFile;
      end;
    //Download if the flag is true
```

```
    if Download then OnNewDownload(Self, DownloadURL, Path,
onNeedsExtracting);
  end;
end;

Procedure TMinecraftUpdater.CheckNatives(var URL, Path: String);
begin

end;



//Downloads the assets
Procedure TMinecraftUpdater.OnDownloadAssetInformation(Sender: TObject;
URL: String);
var
  JSON : ISuperObject;
  Asset : TSuperAvlEntry;
  LegacyPath,ObjectPath, ResourceURL, SavePath: String;
begin
//Loads the correct information about assets
  if Pos('legacy', URL) > 0 then JSON :=
TSuperObject.ParseFile(MinecraftDir + '\assets\indexes\legacy.json',
false)
  else JSON := TSuperObject.ParseFile(MinecraftDir + '\assets\indexes\' +
DVersion + '.json', false);
  LegacyPath := MinecraftDir + '\assets\virtual\legacy\';
  ObjectPath := MinecraftDir + '\assets\objects\';
//Add a download for each asset, and store in relevant directories.
  for Asset in JSON['objects'].AsObject do
  begin
  //Download is at http://resources.download.minecraft.net/<first 2 hex
letters of hash>/<whole hash>
    ResourceURL := ASSETURL + '/' + Copy(Asset.Value.S['hash'], 1, 2) +
'/' + Asset.Value.S['hash'];
    SavePath :=  ObjectPath + Copy(Asset.Value.S['hash'], 1, 2) + '\' +
Asset.Value.S['hash'];
    OnNewDownload(Self, ResourceURL, SavePath, nil);
```

```
  //Now save the same file in virtual/legacy with filename of object.
Download twice is wasteful, but since its only happening once...
    SavePath := LegacyPath + StringReplace(Asset.Name, '/', '\',
[rfReplaceAll]);
  //Only make a copy for the old system 1.7.2 and below
    if DVersion < '1.7.3' then OnNewDownload(Self, ResourceURL, SavePath,
nil);
  end;
end;



//Extracts the URL if its compressed. Always ignore META-INF. Assumes its
a library file
Procedure TMinecraftUpdater.OnCompressedFile(Sender: TObject; URL:
string);
var
  Path : String;
  Index : Integer;
begin
//Try and extract natives
  try
    Index := PosRight('/', URL);
    Path := MinecraftDir + '\versions\' + DVersion + '\natives' +
copy(URL, Index, URL.Length - Index + 1);
    Path := StringReplace(Path, '/', '\', [rfReplaceAll]);
    //Extract to same folder
    TZipFile.ExtractZipFile(Path, ExtractFilePath(Path));
    //Clear META-INF Folder. This may need to change.
    if DirectoryExists(ExtractFilePath(Path) + '\META-INF') then
TDirectory.Delete(ExtractFilePath(Path) + '\META-INF', true);
    //Delete the archive
    TFile.Delete(Path);
    Log.Info('Successfully Extracted ' + Path);
    Except On E:Exception do
    begin
      Log.Error('Unable to extract ' + Path);
      Log.Error(E.ClassName + ':' + E.Message);
```

```
    end;

    end;

end;


{$ENDREGION}

end.
```

UnitSync

```
unit UnitSync;


interface


uses  UnitDownloader,

  System.classes, System.Types, System.SysUtils, FMX.Types, SuperObject,
IOUtils, System.RegularExpressions, Murmur2;


type

  //For if there is a difference between the lists

  TJSONDifferenceEvent = Procedure(Sender : TObject; Path : String) of
Object;


//Generates JSON from files in Minecraft Dir. Attaches Murmur2 Hash for
each file.

  TJSONGenerator = Class

    public

      JSON : ISuperObject;

      Procedure SaveTo(FileName : String);

      Procedure Generate(MinecraftDir : String; ExclusionList :
TStrings);

    private

      MCDir : String;

      Procedure AddFile(FileName : String);

  End;


//Compares the JSON and fires Download and Delete events if there are
differences
```

```
  TFileSyncManager = Class
    public
      MinecraftDir : String;
      RootURL : String;
      OnNewDownload : TNewDownloadEvent;
      OnUpdateLauncher : TStandardEvent;
      Constructor Create;
      Destructor Destroy;
      Procedure Compare;
    private
      Generator : TJSONGenerator;
      ServerJSON : ISuperObject;
      LocalJSON : ISuperObject;
      Procedure OnServerFilesDownload(Sender : TObject; URL : String);
      Procedure CompareJSON(MainJSON : ISuperObject; SubJSON :
ISuperObject; const SObject: TSuperTableString = nil; const OnDifference
: TJSONDifferenceEvent = nil; const path: string = '');
      Procedure OnDownloadFile(Sender : TObject; Path : String);
      Procedure OnDeleteFile(Sender : TObject; Path : String);
  End;
implementation


uses UnitMain;


{$REGION 'Generator'}
//Generate the file listing
Procedure TJSONGenerator.Generate(MinecraftDir : String; ExclusionList :
TStrings);
var
  OnGetFile : TDirectory.TFilterPredicate;
  Files : TStringBuilder;
  Exclusion: string;
begin
  //Don't output to user - incurs far too much overhead
  JSON := SO;
```

```
  MCDir := MinecraftDir;

  //Add defaults

  if Pos('.*\\launcher\\.*', ExclusionList.Text) = 0 then
ExclusionList.Add('.*\\launcher\\.*');

  if Pos('*\\Files.json', ExclusionList.Text) = 0 then
ExclusionList.Add('.*\\Files.json');

  if Pos('*\\versions.json', ExclusionList.Text) = 0 then
ExclusionList.Add('.*\\versions.json');

  //Anonymous Function used to deal with matching

  OnGetFile :=

     Function(const Path: string; const SearchRec: TSearchRec): Boolean

    var

      FileName : String;

      Exclusion : String;

    begin

      FileName := Path + '\' + SearchRec.Name;

      //If the filename matches any of the regexes then do not add it to
the list

        for Exclusion in ExclusionList.ToStringArray do

        if Exclusion.Length > 0 then

          if TRegex.Create(Exclusion).IsMatch(FileName) then

          exit(False);

      //If it doesn't match the Regex, execution continues to here

      //Add the file in JSON form

      AddFile(FileName);

      Result := True;

    end;

  //Enumerates all of the files and passes it to the callback handler

  TDirectory.GetFiles(MinecraftDir, '*', TSearchOption.soAllDirectories,
OnGetFile);

  //Replace all `s with .s and convert back to JSON

  Files := TStringBuilder.Create(JSON.AsJSON).Replace('`','.');

  //Deal with empty directories

  if pos('files', Files.ToString) > 0 then JSON := SO(Files.ToString)

  else JSON['files'] := SO();

  //Add launcher version to JSON
```

```
  JSON.I['launcherVersion'] := launcherVersion;
  //Add exclusion list to JSON
  JSON['exclusionList'] := TSuperObject.Create(stArray);
  JSON['exclusionList'].AsArray.Add('.*\\launcher\\.*');
  for Exclusion in ExclusionList do
JSON['exclusionList'].AsArray.Add(Exclusion);
  FreeAndNil(Files);
end;

//Adds file to JSON and dynamically create objects
Procedure TJSONGenerator.AddFile(FileName : String);
var
  ObjectPath : TStringBuilder;
begin
  //Use TStringBuilder for replace performance. Makes a difference when
indexing large amount of files. StringReplace() is too slow.
  ObjectPath := TStringBuilder.Create(FileName);
  //Strip out MinecraftDir and change it to files\ in one shot
  ObjectPath.Replace(MCDir, 'files');
  //Mark existing dots to be converted back later by replacing it with `
  ObjectPath.Replace('.','`');
  //Now convert \ into .
  ObjectPath.Replace('\', '.');
  //Create all the objects in the path
  JSON.ForcePath(ObjectPath.ToString, stObject);
  //Then add the Murmur2 hash of the file. 90% of time taken up by I/O
  try
    try
      JSON.I[ObjectPath.ToString] := CalcMurmur2(FileName);
    Except On E:Exception do log.Error('Unable to add file ' + FileName);
    end;
  finally
    FreeAndNil(ObjectPath);
  end;
end;
```

```
//Save the JSON to filename
Procedure TJSONGenerator.SaveTo(FileName : String);
begin
  try
    //Save the file
    JSON.SaveTo(Filename, True, True);
    Log.Info('Saved JSON to ' + FileName);
  Except On E:Exception do
  begin
    Log.Error('Failed to Save to ' + Filename);
    Log.Error(E.ClassName + ':' + E.Message);
  end;
  end;
end;

{$ENDREGION}

{$REGION 'File Syncer'}
constructor TFileSyncManager.Create;
begin
  inherited Create;
  Generator := TJSONGenerator.Create;
end;

destructor TFileSyncManager.Destroy;
begin
  Generator.Destroy;
  inherited Destroy;
end;

//Generates local file list and downloads remote. Fires
OnServerFilesDownload when the files.json has been downloaded.
Procedure TFileSyncManager.Compare;
```

```
begin

  //Download Server's Files.json

  if Assigned(OnNewDownload) then OnNewDownload(Self, RootURL +
'/Files.json', MinecraftDir + '\ServerFiles.json',
OnServerFilesDownload);

end;


//Actually instigate the download and deletion of files

Procedure TFileSyncManager.OnServerFilesDownload(Sender: TObject; URL:
String);

var

  ExclusionList : TStringList;

  Exclusion : ISuperObject;

begin

  try

    //Acquire ExclusionList from Server's JSON

    ExclusionList := TStringList.Create;

    ServerJSON := TSuperObject.ParseFile(MinecraftDir +
'\ServerFiles.json', False);

    //Check if the launcher has been updated

    if ServerJSON.I['launcherVersion'] > launcherVersion then

    begin

      onUpdateLauncher(Self);

      //Do not allow further execution

    end;

    for Exclusion in ServerJSON['exclusionList'] do
ExclusionList.Add(Exclusion.AsString);

    //Ensure these files are not downloaded

    ExclusionList.Add('.*ServerFiles.json');

    ExclusionList.Add('.*\\launcher\\.*');

    ExclusionList.Add('.*\\versions.json');

    ExclusionList.Add('.*\\Files.json');

    //Generate File list

    Generator.Generate(MinecraftDir, ExclusionList);

    ExclusionList.Destroy;
```

```
    //Prepare JSON - replace . with ` so that it doesnt confuse . with a
path and only look in files
    LocalJSON := SO(StringReplace(Generator.JSON['files'].AsJSON, '.',
'`', [rfReplaceAll]));
    ServerJSON := SO(StringReplace(ServerJSON['files'].AsJSON, '.', '`',
[rfReplaceAll]));
    //Compare LocalJSON AGAINST ServerJSON, and anything on the
ServerJSON that is not on LocalJSON will be downloaded
    CompareJSON(LocalJSON, ServerJSON, ServerJSON.AsObject,
OnDownloadFile);
    //Do the opposite. Delete anything not on ServerJSON
    CompareJSON(ServerJSON, LocalJSON, LocalJSON.AsObject, OnDeleteFile);
    //Update the JSON by using server copy
    if FileExists(MinecraftDir + '\Files.json') then
      TFile.Delete(MinecraftDir + '\Files.json');
    TFile.Move(MinecraftDir + '\ServerFiles.json',  MinecraftDir +
'\Files.json');
  Except On E:Exception do
  begin
    Log.Error('Unable to sync mods. Minecraft may not work properly.');
    Log.Error(E.ClassName + ':' + E.Message);
  end;
  end;
end;

//Delete file as long as it is not on the exclusion list
Procedure TFileSyncManager.OnDeleteFile(Sender: TObject; Path: String);
var
  FileName : String;
begin
  try
    //Replace . with \
    FileName := StringReplace(Path, '.', '\', [rfReplaceAll]);
    //Replace ` with .
    FileName := StringReplace(FileName, '`', '.', [rfReplaceAll]);
    //Replace files with MCDir
```

```
    FileName := MinecraftDir + '\' + FileName;
    log.Warn('Deleting ' + FileName);
    TFile.Delete(FileName);
  Except On E:Exception do
  begin
    Log.Error('Unable to delete ' + FileName);
    Log.Error(E.ClassName + ':' + E.Message);
  end;
  end;
end;

//Download the file and handle launcher updates differently
Procedure TFileSyncManager.OnDownloadFile(Sender: TObject; Path: String);
var
  URL, FileName : String;
begin
  //Update file as normal
    //Replace .s with /
    URL := StringReplace(Path, '.','/', [rfReplaceAll]);
    //Replace ` with .
    URL := StringReplace(URL, '`','.', [rfReplaceAll]);
    //Get Path by converting to backslashes
    FileName := StringReplace(URL , '/', '\', [rfReplaceAll]);
    //Insert relevant Root URL and MCDir
    FileName := MinecraftDir + '\' + FileName;
    URL := RootURL + '/' + URL;
    OnNewDownload(Self, URL, FileName, nil);
end;

//Recursive subrouting to traverse and compare JSON lists
Procedure TFileSyncManager.CompareJSON(MainJSON : ISuperObject; SubJSON :
ISuperObject; const SObject: TSuperTableString = nil; const OnDifference
: TJSONDifferenceEvent = nil; const path: string = '');
var
  Keys, Items, Item, ArrayItem: ISuperObject;
```

```
  Key: string;
  i: Integer;
  Extract : Boolean;
begin
  Extract := False;
  //Check JSON was given to compare against
  if Assigned(SObject) then
  begin
    //Get all the names of the keys, and their values
    Keys := SObject.GetNames;
    Items := SObject.GetValues;
    //Iterate through the keys
    for i := 0 to Items.AsArray.Length - 1 do
    begin
      //Get all the sub keys and values
      Key := Keys.AsArray[i].AsString;
      Item := Items.AsArray[i];
      // If key exists in the MainJSON
      if Assigned(MainJSON[Path + Key]) then
        // But the values (hashes) of the keys and subkeys are not equal
then continue to compare
        if not(LowerCase(SubJSON[path + Key].AsString) =
LowerCase(MainJSON[path + Key].AsString))
          then Extract := True
        else //Drop out of else
      //otherwise it is not present on the other list so extract anyway
      else Extract := True;
      if Extract then
        // If object, recursive call
        if Item.DataType = stObject
          then CompareJSON(MainJSON, SubJSON, Item.AsObject,
OnDifference,  Path + Key + '.')
        // Same with array except iterate through each item
        else
          if Item.DataType = stArray then
```

```
            for ArrayItem in Item do
              CompareJSON(MainJSON, SubJSON, Item.AsObject, OnDifference,
Path + Key + '.')
          // Otherwise it is a raw datatype, and are not equal, so fire
the OnDifference Event to take appropriate action
        else
          begin
            OnDifference(Self, Path + Key);
            //Delete so that if it is run again in reverse, no duplicates
for already processed items
            MainJSON.Delete(Path + Key);
          end;
        Extract := False;
      end;
    end;
end;

{$ENDREGION}

end.
```

Appraisal

The table below analyses each original objective to establish if and how they have been met.

No	Objective	Met	Comments
1	The system must be able to launch Minecraft, using the Java Runtime Environment	Yes	This has very clearly been achieved, as the testing has showed, that Minecraft is launched directly from the launcher. This may not hold true for future version of Minecraft however, as new parameters are added, the launcher may also need to be updated to accommodate for these changes
2	The system must identify changes between the repository's version of Minecraft and the student's copy of Minecraft, when the student runs the system	Yes	The comparison of two different file listings is done very efficiently by recursively traversing both lists as needed, and comparing hashes. However, the longest part of the process is reading the current files from the hard disk and calculating its corresponding hash. The objective has been met, but there is room for improvement
3	The system must then download any changes; ideally in a pooled, multithreaded format, and transform the Minecraft directory such that it reflects the repository's state	Yes	There are 4 threads being used in the current launcher to download files. This is more appropriate on the school network, where download speeds are high and more threads can be utilised. The thread number can be configured at design time. The launcher is successful in downloading any changes, and reflecting the current state of the repository by deleting any unwanted files
4	The system should recognise the teacher is logged in and present different options accordingly	Yes	The launcher presents different panes, and does not limit the teacher by a time quota. These different panes have hover-over popups that inform the teacher of their functionality. If the database has not been configured, the database pane is not shown. Students can only see the log pane and requests pane, as originally planned
5	The system should store a log of events that occur, locally in the Minecraft directory	Yes	The launcher saves a "log.txt" in the same directory of the launcher executable, when the launcher closes. Although this may not be the Minecraft directory in every case, following the prescribed user guide will lead to this configuration. Additionally, these same messages are logged to the log pane, which both students and teachers have access to, so should there be a reading/writing issue to the directory, messages can appear there. They are also tagged with [Information], [Warning], or [Error] tags, where the latter two are colour coded orange and red respectively

6	The system should allow the teacher to copy their version of Minecraft to the repository as the working copy, being able to exclude files with a regex	Yes	The sync pane allows for this objective to be met thoroughly. There is an input box that allows for the entry of several regular expressions. The pane also provides an optional automatic copy function to meet the first half of this objective.
7	The teacher should be able to upgrade/downgrade available Minecraft versions within the system	Yes	The updates pane facilitates this by downloading a list of all versions that are 1.6.4 and above. Although the teacher cannot downgrade below 1.6.4, the launcher has been tested to work up to Minecraft 1.8, with no versions in between having any issues
8	The system should be able to handle aliases between a school user account and Minecraft name	Yes	Their teacher-set alias is automatically displayed to them in the username box when the start the launcher, and is used in game. Therefore this objective has been met completely
9	The system should have an alias request feature or mod request feature which notifies the teacher	Yes	The request pane for students allows them to ask the teacher for an alias, who can accept or decline the request via the database pane. The student can also request mods, which the teacher can view in the database pane. The teacher is not notified in the sense of a live update, but is able to view any new requests via the database pane, so this fulfils the objective
10	The teacher should be able to modify the parameters for launching the Java Virtual Machine	Yes	The Java tab under the settings pane allows the teacher for modifying the maximum and minimum RAM parameters, used by the JVM. These are the most commonly modified parameters, and although the objective has been met, there is scope for extension here
11	Log off and log on times should be stored and retrievable	Yes	The database pane allows the teacher to view log off times, log on times, and the computer which the user logged on to. This is known as a single session. A session begins at the moment the user clicks the play button, and ends when the Minecraft window has closed
12	The teacher should be able to set a time quota that students are able to play Minecraft for	Yes	The teacher is able to right click users, under the user tab in the database pane, and modify either the (daily) global time quota, or a specific user's time quota remaining. This meets the objective
13	The system should log all messages sent by the user so the teacher can review incidents	Yes	The messages tab under the database pane provides the user that sent the message, the date, and the actual message, as the objective proposed

14	The teacher should be able to ban students	Yes	The teacher can simply right click the user under the users tab in the database pane, and click ban, or unban. The user is instantly banned, and if they are playing Minecraft, their session is terminated, and the banned message appears. They are no longer able to play Minecraft, as intended
15	Messages, users and session information should all be searchable	Yes	Each of the tabs under the database pane has a search box, which searches the database as entry occurs. This allows for quick and efficient searching. Dates are also searchable. All data is partially matched, using the SQL like operator. Therefore, the objective has been met
16	The system should present some statistics to the teacher, such as the average time per user	Yes	Under the users tab in the database pane, each user is presented with 2 additional columns – session count, and average time spent. These are calculated based on the session data. More statistics could be included however.
17	The system should be able to save backups of a Minecraft installation, and restore them	Yes	The backup tab under the settings pane provides 2 buttons – restore backup and save backup. Each of these buttons presents a file dialog that allows the teacher to save/open a ZIP file created by the launcher, containing the contents of the Minecraft directory. The launcher is able to un-zip these files to the Minecraft directory

All objectives were met, and the majority were implemented to a higher level than specified by the objective. Occasionally, improvements could be made, however these were not huge issues.

Feedback

Client Feedback Letter

The following letter was received from Mr Macleod, after I demonstrated the launcher to him and presented him with the user guide:

23rd April 2015

Dear Harjot,

Thank you for demonstrating your Minecraft management software suite yesterday.

Overall I was very impressed both with the functionality and the presentation of the software. At present the synchronisation of server and client software, particularly the mod files, is tricky and error prone. I'm therefore very pleased to have a tool which simplifies the process. The ability to track user log-on times is particularly welcome and I'm not aware of any other software which can do this. It will make it easier to make Minecraft available in the school and remove some of the objections that other teachers offer.

The overall program is clearly very complex and it will take some time to master. Your user manual looks comprehensive and useful here, it seems well laid out but it will take some time for me to work my way through it.

We did encounter some difficulties making the software run initially, this seems to be a configuration issue on the office machines and I'm not sure that this is completely resolved. It may be that you need to assist here making sure the program can run where it is needed.

At present server files and mod files need to be manually uploaded to the server to become part of the system. That is what we do at present so it does not represent an extra burden, but as we discussed it would be a worthwhile improvement to allow this aspect of management to be automated.

The facility to back up and restore the Minecraft setup is not something we've used yet but looks very useful in the event of problems with the server.

The overall look and feel of the program is delightful, you are to be congratulated on the work that you've clearly put into it. I do believe that it may have commercial potential, I'm sure I'm not the only teacher struggling with the management of this complex game.

Yours sincerely

Graham Macleod

Head of Academic Computing

A signed copy of this letter will be attached to this document

Feedback Analysis

The feedback was generally very positive.

Mr Macleod mentions in the first paragraph that he is pleased that he is able to use the launcher to simplify the syncing files process, which corresponds to objective 2. He highlighted the issue that in his current system, it is extremely trick and error prone, which proves that my analysis was correct, and the implementation of the proposed objective solved the problem suitably.

In the same 1st paragraph, he mentions that tracking user log-on times is a welcome feature, since there is no other software that can do this. Having a history of access times appears to give teachers confidence in the usage of Minecraft at school, and cause less issues of game related objections. Log on and log off time tracking are covered by objective 11, and clearly Mr Macleod approves of the implementation of it.

In the 2nd paragraph, Mr Macleod describes that the program appears to be very complex, but the user guide should guide him through it satisfactorily. The perceived complexity does raise a small concern over whether the design was correct, or whether the complexity of usage simply cannot be helped.

Mr Macleod, in the 3rd paragraph, explains that there were issues running the Minecraft launcher on the school machines. The issue has been isolated, and was in fact due to a compilation error, where a copy of the application was accidentally compiled under the wrong settings in Delphi, and then distributed to Mr Macleod unknowingly. A corrected copy has been built and sent to Mr Macleod.

The 4th paragraph touches on some general improvements, most notably to the synchronisation process, where he suggests that files could potentially by automatically uploaded to the server without him having to manually copy them there. This would involve extending objective 6.

Paragraph 5 compliments the backup and restore feature, asserted by objective 17. This appears to be a less commonly used feature.

The last paragraph shows Mr Macleod is happy with the design. It is extremely interesting to note that he thinks it has potential as a commercial product, and could in fact be helping other schools solve the same issues that are associated with managing Minecraft in a school environment.

Future Extension

There were some improvements identified in the analysis of the objectives, as well as the one suggested by Mr Macleod:

1. Automatically sync files to the server. This can be achieved in a number of ways, but considering that a HTTP server is already running on the file repository server, it would be most convenient installation-wise for the user if a PHP script was written, that the launcher could use to upload files. In practice, this strategy would need to be tested and refined, as multithreaded uploading may be required to reduce the wait time significantly – Minecraft consists of thousands of small files, rather than less, but larger ones. This feature would enhance objective 6 significantly.
2. Alternatives behind the syncing process could be implemented. Currently, all files are read, and Murmur2 hashes corresponding to each file are indexed. This can become grossly inefficient with large numbers of files, and is the biggest bottleneck in the application start up time at the moment. Perhaps a quicker check could occur at application launch, on file versions, and if these are different, continue with the slower file hash comparison. This is with regard to objective 2, which was met. Perhaps an external version control system (VCS) such as Subversion could be embedded, such that the launcher does not handle file changes, but the external VCS application does.
3. More statistics could be presented in the launcher. This is simply a matter of acquiring some time to write more SQL queries. Perhaps the launcher could even provide the ability for writing and saving custom SQL queries that can be executed within the launcher. This would provide immense flexibility, but presents some basic security and user reliability issues, and so safeguards against deleting tables or accidental data may need to be implemented. This would improve the result of objective 16.
4. The launcher could evolve into a tool that is also able to manage servers. This would give teachers a unified interface, rather than having to log on to a separate machine or console to access the Minecraft server and make changes. This would involve writing a server application that would run on the Minecraft server machine, and then execute commands on behalf of the machine.