PowerGrid documentation

An overview of the PowerGrid project, how it works, and what the differences are with other Runescape bot clients.



**Table of Contents**

[PowerGrid documentation 1](#_Toc376726397)

[Chapter 1 – Introduction 3](#_Toc376726398)

[Chapter 2 – Overview of PowerGrid 4](#_Toc376726399)

[2.1 – Communication with the Runescape client 4](#_Toc376726400)

[2.2 – Cross-platform Graphical User Interfaces 4](#_Toc376726401)

[2.3 – High-level task descriptions in favor of direct control 4](#_Toc376726402)

[Chapter 3 – Structural overview 5](#_Toc376726403)

[3.1 – Structural diagram of PowerGrid 5](#_Toc376726404)

[3.2 – Summary of each of the modules in PowerGrid 6](#_Toc376726405)

[3.3 – The process of performing tasks in PowerGrid 7](#_Toc376726406)

[Chapter 4 – Implementation 8](#_Toc376726407)

[Chapter 5 – Legal notes 9](#_Toc376726408)

# Chapter 1 – Introduction

Many bot clients already exist for the MMORPG Runescape. However, since the introduction of Runescape 3, most of these bot clients have been rendered useless, as using these bot clients may sometimes cause the player to be banned from the game. As such, only few bot clients are able to provide the functionality that is desired from a bot client without the implied risk of being banned.

The above problems, together with observations of the points where other bot clients failed on, calls for a new approach to Runescape bot development. This project aims to fill the gap left behind by the other bot clients after the Runescape 3 update, and to provide the functionality requested by its users more than other bot clients did before. This project aims to do so by making the client completely open-source. Also, to prevent direct manipulation of the Runescape client, this new bot client will not perform any byte code injection on the running Runescape client. Instead, the core of the bot client is written in C++, an object-oriented language which is somewhat similar to Java. Because of this, most of the bot client’s botting functionality will happen completely separate from the Java Virtual Machine, and as such is invisible from the Runescape client.

Putting all the above together, we present you PowerGrid. A revolutionary, open-source bot client that intends to make life easier for everyone by providing functionality that automatically reads the data from the Runescape client and presents it as native C++ objects. Because of this, the bot can provide at least the same functionality as traditional injection bots, but without modifying the game data.

The final goal of PowerGrid is to provide users with a tool that can play Runescape completely by itself, automatically deciding on the tasks to perform based on changes in the environment. PowerGrid will even be able to perform abstract tasks like leveling a certain skill to a certain level, or get to a certain destination. PowerGrid should then automatically decide on the concrete tasks (what methods to use for quick travelling, or what method to use to train the requested skill), based on the efficiency of each task.

PowerGrid will not contain all this functionality by itself, but will present a platform that allows others to add this functionality to PowerGrid by developing plugins for it. Such plugins can be added to and removed from PowerGrid at any time by the user, allowing each user to add exactly the functionality he or she desires.

# Chapter 2 – Overview of PowerGrid

## 2.1 – Communication with the Runescape client

The main component of any Runescape bot client is getting the information from the Runescape client, and sending actions back to the client. To achieve this between the C++ bot and the Java client, it is required to use an intermediate system provided by the Java Virtual Machine called JNI (Java Native Interface). A drawback of the JNI is, that it is quite a basic system, and not at all suitable for the intensive and complex operations that a bot client requires. Therefore, PowerGrid uses a library that extends JNI and provides a more natural way to communicate with the Java Virtual Machine. This library is called JACE, and it provides a way to communicate with Java classes and objects as if they were C++ classes and objects, making it easier to bridge the gap between the two languages. Because of the way JNI and JACE work, this setup also allows PowerGrid to read data from the Runescape client naturally without modifying it in any way. JACE has been modified to work with the obfuscation of the Runescape client, presenting all data from the client using natural names.

## 2.2 – Cross-platform Graphical User Interfaces

PowerGrid is written in C++, and as such, it is inherently platform-dependent. To make it easier to use PowerGrid on multiple platforms, such as Windows, Mac OS, and Linux, PowerGrid makes use of a sophisticated library that aims to remove all platform differences, allowing the same code to work on almost all popular platforms. This library (called Qt) also features ways to dynamically load plugins, something which will extend PowerGrid with additional features and functionality. Qt also provides many convenient tools and components that make it relatively easy to develop for PowerGrid.

## 2.3 – High-level task descriptions in favor of direct control

PowerGrid is meant to be used with high level tasks, so for example “train mining skill to level 99”, instead of “mine this much of this type of ore at that place”. PowerGrid is then meant to automatically decide the best location and ore type between the options it knows about. This is meant to result in a more natural playing style, since PowerGrid can decide to change strategies midway and go to a different mining spot if that turns out better (for example, when the initial mining spot is very crowded). By giving high-level tasks to PowerGrid, it may be possible to use strategies that players would normally not think of, but are more efficient than conventional strategies, while still remaining more player-like due to this dynamic behavior.

# Chapter 3 – Structural overview

This chapter provides an overview of how the PowerGrid client works. This chapter attempts to illustrate this by examples understandable by any Runescape player. It is not needed to understand everything in this chapter in order to use PowerGrid, but it is recommended to read through it in order to understand what PowerGrid does to ‘play’ Runescape.

## 3.1 – Structural diagram of PowerGrid

PowerGrid consists of many small parts, but these can be grouped together in only a few large parts with distinct roles. These groups are chosen in such a way that there is a clear purpose for each module. By presenting PowerGrid in this way, it becomes easier for everyone to understand the way information flows between the different modules in PowerGrid and the Runescape client. The diagram below shows this flow of information between all parts involved in PowerGrid.



Figure 1 - Structural Diagram of PowerGrid

## 3.2 – Summary of each of the modules in PowerGrid

As seen in the diagram in chapter 3.1, PowerGrid consists of a few large modules. In this chapter, a summary of the tasks of each of these modules is given.

RS Client

The Runescape client, running in the Java Virtual Machine. It connects to the Runescape servers to allow player to play the game online. PowerGrid can read data from the Runescape client which it uses to decide its actions.

PG Loader

The PowerGrid client loader, which loads and starts the Runescape client. It is built with the focus on performance. As such, it is able to start the client faster than almost any other client, including the official Runescape Launcher. It can also be used as a stand-alone client that only launches the Runescape client without any bot functionality enabled.

JACE

The library that is used to communicate with the Java Virtual Machine. Through JACE, the bot can read information from the client, and take actions based on this. It serves as the main component linking the Java Virtual Machine and the PowerGrid C++ client. This module provides access to the data from the Runescape client, and can send input commands back into the client to allow the PowerGrid AI to ‘play’ Runescape.

PowerGrid AI

The PowerGrid AI is the core of the bot. This component deals with executing tasks and performing the actual botting. It reads data and sends back commands using JACE. Users of PowerGrid can give instructions to the AI using the GUI. The PowerGrid AI can also use additional functionality provided by the Plugin System.

Plugin System

The part of PowerGrid that loads external plugins and makes them available to the PowerGrid AI. It’s main purpose is to allow other developers to write extensions for PowerGrid to improve or add functionality. The PowerGrid AI will use this added functionality when needed.

PowerGrid GUI

The graphical user interface of PowerGrid. It allows users to select tasks for PowerGrid to perform, and have direct control over the PowerGrid client. The GUI is presented next to the Runescape client in a separate window. This is because Java (AWT/Swing) GUI parts and C++ (Qt) GUI parts cannot be mixed.

## 3.3 – The process of performing tasks in PowerGrid

To further illustrate what PowerGrid does and how it works, an example is given here of a user having PowerGrid execute a simple task.

Imagine the user wants to train the mining skill level of his/her character. If PowerGrid contains functionality to train this skill, then the user can select the appropriate skill in the list of skills that can be trained. PowerGrid will then look for tasks it knows that promise to train the skill (in this case: mining). PowerGrid will compare the skills-improving tasks based on efficiency and execute one or more of these tasks in sequence to achieve the goal the user has set (which can be in terms of experience gained, or reaching a certain level, amongst others).

To execute the task, PowerGrid may need to execute smaller subtasks (like moving to a specific spot, or collecting a specific item from the bank). In case of mining, PowerGrid would need to move the character to a mining spot, then repeatedly click minable rocks to collect ore, and (possibly) go to the bank and back when the character’s inventory is full. These are all subtasks, that are provided either by PowerGrid itself, or are included in one of the plugins loaded by PowerGrid. The task that involves moving to a mining spot may, in turn, also consist of smaller tasks, like using a lodestone, teleport, or just walking there. This way, large and complex tasks can be done by combining smaller and easier tasks.

Over time, PowerGrid may be able to support a large variety of tasks that help it to execute extremely complex tasks, like fighting against enemies in a group together with other players. PowerGrid may with time even be able to communicate with other bot clients and optimize task execution by telling each other about what they’re seeing. Consider the previous example of mining. Imagine that at a certain mining spot there are other users using PowerGrid. Then these clients may be able to tell that the mining spot they’re in is already very crowded. As such they may be able to pass on this information to other PowerGrid clients, so that these clients choose a different mining spot. This helps to prevent the PowerGrid client from navigating to a mining spot, only to find that it is too crowded when it gets there.

# Chapter 4 – Implementation

This chapter will go deeper into the implementation of all the parts of PowerGrid. It illustrates how the PowerGrid client does what it does. This chapter is mostly intended for programmers and developers, as it may contain terminology specific to programming or Runescape bot development.

## 4.1 The PowerGrid loader

The PowerGrid loader is the only part in PowerGrid that is written in Java. It’s only purpose is loading the Runescape client as quickly as possible. For this purpose, the loader is able to cache the client configuration and client data it needs, instead of re-downloading this data every time the client is started.

This caching behavior depends on a characteristic part of the configuration data to determine if there is a new version of the Runescape client (if there is, the loader has to download the client anyway). By analysing this characteristic part, the PowerGrid loader is able to determine if there is a new Runescape client version with less than error rate.

The procedure for loading the Runescape client does not depend on the runescape website to navigate to the page where the applet is loaded. Instead it uses the same technique as the open source official Runescape loader. The official loader requests a special config file from the Runescape servers that presents the data required for downloading and launching the Runescape client in a format easily read by a program. The PowerGrid loader can take the URL for downloading the client data as well as the parameters required for launching it directly from that file. This results in faster loading times, since only one request has to be made to the Runescape servers before we can start the client (as opposed to navigating the Runescape website, which requires at least 2 page loads on top of parsing the html that is returned by the Runescape web server).

## 4.2 JACE and the JNI

For reading data from the Runescape client and giving instructions to it, PowerGrid makes use of a library called JACE. The library is built statically and directly included in the PowerGrid application.

JACE was originally written by Toby Reyelts, who licensed it under a BSD license. The library is modified and optimized for PowerGrid. In order to support a more intuitive way of loading Runescape data, changes are made to JACE so that we can dynamically map C++ classes and objects to their respective Java counterparts, even though the Java classes are obfuscated. This behavior is similar to how injection clients present the client’s classes, with the difference that those clients have to modify the client data in order to present it that way. JACE links C++ proxy classes to Java classes as they are loaded, and whether or not that class is obfuscated does not cause any additional overhead. As such PowerGrid can provide the same functionality that injection clients provide in the same way, but without directly modifying the client.

## 4.3 The Entity Framework

This module provides a framework for storing data and executing operations on that data in a unified way. It provides a platform that is both very dynamic and in practise relatively fast. Data is represented as an instance of the Entity class, and properties of the data are assigned to the Entity in the form of Components.

Components represent small, but meaningful bits of data about a certain Entity. For example, PowerGrid provides some basic Components like ID, Name, and Position. These Components can be combined to form Entities with any combination of properties, and it also enables plugin developers to iterate over all Entities with a certain property.

The true power of the Entity system is shown in the following example: We again take the (relatively) simple example of a miner plugin. The miner plugin provides a Classifier to PowerGrid that assignes a Component to ore Entities. It does so by checking the Entity’s ID component, and comparing the id with known ids of ore types. The component that is assigned is a component contained in the plugin, named Ore. The Ore component contains a field that indicates the type of Ore that can be mined from that Ore Entity. The miner Plugin also provides a Task that mines the nearest Ore Entity. For this purpose, we ask the World for a list of all Ore type Entities. That list can be acquired in constant time, making these kinds of operations much faster than similar operations with other bot clients. We then find the nearest Entity from that list (that functionality is contained in the PowerGrid API), so all operations uptil now can be done in two statements:

API::Navigator\* nav;

Entity\* e = API::Math::minDist(nav->currentPosition(),world->getAll<Ore>());

Now the Entity e contains the nearest Entity with an Ore component. All that’s left to do is move towards the Entity and click it. Moving can be done using the Navigator class from the PowerGrid API (already instantiated above):

nav->navigate(e->get<Position>(), API::Navigator::InView);

The API::Navigator::InView parameter hints that we only need to have the target in our view, and don’t need to move to exactly this spot. The navigation will happen in the background, so we will have to wait for it to complete. This can be done asynchronously using the TaskContext of our task:

context()->waitUntil(&API::Player::idle);

This function call waits until API::Player::idle() returns true. It does so in the background, waiting for updates in the World and then repeatedly checking the condition until it becomes true. It is implemented in a thread-safe way using Qt’s signal/slot mechanism, and is generally more efficient that waiting in a loop.

Lastly, we click the Ore Entity we moved towards using the API::Mouse class, and wait for the player to become idle:

API::Mouse::click(e);

context()->waitUntil(&API::Player::idle);

This concludes the Mining task. This task can be repeated indefinitely to mine rocks in the Runescape World. The final body of the Task’s execute function is as follows:

API::Navigator\* nav;

Entity\* e = API::Math::minDist(nav->currentPosition(),world->getAll<Ore>());

nav->navigate(e->get<Position>(), API::Navigator::InView);

context()->waitUntil(&API::Player::idle);

API::Mouse::click(e);

context()->waitUntil(&API::Player::idle);

As can be seen, the task’s body is relatively short and concise.

## 4.4 The Plugin System

This module provides the functionality required to dynamically load additional features and improvements to provide users with a way to extend their experience and challenge developers to help make PowerGrid better. In order to stay cross-platform, PowerGrid again makes use of a Qt class (QtPluginLoader) to ensure these plugins work on all supported platforms. There is, however, one limitation to using native code when dealing with plugins in this way: A plugin written for PowerGrid has to be compiled for various platforms in order to use it for that platform.

This is a recurring problem that cannot be overcome. Instead, we encourage plugin developers to make their plugins open-source so that anyone can compile a version of the plugin for any supported platform if they wish to.

Plugins are represented by a single class in the plugin library, which is a subclass of the (pure virtual) class PGPlugin. Plugins implement the member functions declared by PGPlugin to provide PowerGrid with the contents of the plugin. Plugins may provide PowerGrid with tasks (also known as scripts for other bot clients), classifiers (small pieces of code that aim to identify data from the Runescape client), skill trainers (tasks that train a certain skill) as well as the name of the plugin, its authors, and a description.

## 4.5 The PowerGrid GUI

This module presents a graphical user interface for use with PowerGrid. It provides information about the status of PowerGrid and its settings. It also allows users to select and configure tasks, after which the PowerGrid AI will execute them.

The GUI is built using the QtWidgets library, which is a part of the Qt framework. A tool provided by Qt called Qt Designer is used for designing the GUI. Because of using this framework, the PowerGrid user interface can look the same on all platforms supported by Qt (It should however be noted that not all platforms supported by Qt are also supported by PowerGrid).

< This module is also still being worked on and features are added when corresponding features are implemented in the other modules. For now, no conclusive information can be provided about the layout of the GUI or its individual components. >

# Chapter 5 – Legal notes

It is important to state the legal information surrounding the project (don’t worry, it’s only one page), since this project deals with a variety of potential legal complications.

First of all, PowerGrid is completely free and open-source, and is licensed under GNU GPL version 3. A copy of this license is available at <http://www.gnu.org/licenses/gpl.html>. A copy of this license is also shipped along with PowerGrid. Additionally, on the PowerGrid Git repository, the terms of the GNU GPL version 3 are also described in the file named “COPYING”.

Furthermore, we, the authors of this document, and the main developers of PowerGrid, have no connection to Jagex Ltd. or Runescape whatsoever. Any and all trademarks and/or rights are property of their respective owners.

The libraries that are used for PowerGrid may fall under different licenses. See the documentation for these libraries for the license on that library.

If you want to develop plugins for PowerGrid, you need to be aware of the legal concerns surrounding these activities. These concerns are stated below:

For developing plugins, the following terms (and suggestions) apply:

* + The plugins you release must be licensed under the GNU GPL (version 3 or later), or a GPL-compatible license. The terms of this license must be followed when you distribute your plugin.
  + You should be aware of the exception to the GPL that applies to PowerGrid. This exception allows linking with the JNI and to the Runescape client (which are both not considered free software).
  + You must honor the licenses that apply to PowerGrid and any libraries it uses, including the licenses for JACE and Qt.
  + According to the GPL license, you must either distribute your source code along with the plugin, or be willing to provide it when people ask for it.
  + You need to place copyright on your plugin to license it under the GPL. For this purpose, please visit <http://www.gnu.org/licenses/gpl-howto.html> and follow the instructions there to apply the GPL to your plugin.
* For working on PowerGrid itself, the same terms apply. However, there are some further legal issues and concerns surrounding developing for PowerGrid itself, which exceeds the scope of this document. Please contact Chronio for information and advise surrounding these concerns.

Note that these terms are not here to restrict you, but rather to protect your freedom as a developer. By sticking to these rules, you can protect yourself and your software from copyright infringement, and it also helps others when you share your work. Of course, the inverse is also true: you can also read the work of others and use it to improve your own work.