Matt Fuqua

fuqua.matt@gmail.com

Buzz Bot

Technical Specification

Contents

[I. Abstract 1](#_Toc48815086)

[II. Definition of Key Terms 1](#_Toc48815087)

[III. EPGP Overview 1](#_Toc48815088)

[A. Relative Gear Priority Problem 1](#_Toc48815089)

[B. EPGP Equations 2](#_Toc48815090)

[C. The “Hoarding” problem 2](#_Toc48815091)

[D. The “New Raider” problem 3](#_Toc48815092)

[IV. BuzzBot Technical Overview 4](#_Toc48815093)

[V. Raid Module 4](#_Toc48815094)

[A. Raid Templates 5](#_Toc48815095)

[B. Raid Events 6](#_Toc48815096)

[VI. EPGP Module 9](#_Toc48815097)

[A. Configuration 10](#_Toc48815098)

[1. EP Minimum 10](#_Toc48815099)

[2. GP Minimum 10](#_Toc48815100)

[3. Decay Percentage 10](#_Toc48815101)

[4. Decay Day of Week 10](#_Toc48815102)

[B. Adding Users 11](#_Toc48815103)

[C. Switching Active Aliases 12](#_Toc48815104)

[D. Awarding EP and GP 12](#_Toc48815105)

[E. Determining User Priority 13](#_Toc48815106)

[F. Auditing Users 14](#_Toc48815107)

[G. Removing EPGP Transactions 16](#_Toc48815108)

[VII. Software Overview 17](#_Toc48815109)

[A. Core Services 17](#_Toc48815110)

[1. EPGP Service 17](#_Toc48815111)

# Abstract

BuzzBot is an administrative service that controls a “bot user” on the Discord VOIP platform. The purpose of the service is to provide utility and automation of common functions for a Classic World of Warcraft gaming organization. The bot interacts with users through Discord directly, using the Discord.Net communication library. BuzzBot also integrates third party RESTful services to help support its key functions.z

# Definition of Key Terms

**Buzz** – Name of the Classic World of Warcraft gaming organization for which the bot was written. Named for the mascot of the Georgia Institute of Technology.

**WoW** – Acronym for World of Warcraft

**Raid –** A term for an organized gaming event where 20-40 users will work together to achieve shared game objectives.

**Loot –** A term for in-game rewards distributed for achieving game objectives.

**EPGP –** Acronym for Effort Points – Gear Points, an accounting philosophy for distributing in-game rewards. Effort points are awarded to members for helping achieve in-game team goals, and gear points are given whenever they are awarded a piece of loot. The ratio of effort points over gear points determines loot priority.

**PR –** Shorthand for “priority”, a user’s relative standing within the EPGP system.

**Guild** – A term for a chartered gaming organization within the World of Warcraft ecosystem.

# EPGP Overview

EPGP is the main accounting philosophy that drives reward distribution within the Buzz guild. Participation in raids and other shared guild objectives will allow users to accrue effort points, while acquiring loot will result in accrual of gear points. The ratio of effort points to gear points determines a users priority (ref Equation 1).

Equation 1. Priority Equation

## Relative Gear Priority Problem

In WoW, loot desirability can be loosely determined by three key factors:

* Quality
* Purpose
* Power

These three factors are defined more specifically in the game as rarity, slot, and item-level respectively. Because loot desirability is variable based on these factors, GP can not be assigned as a static value for individual pieces of loot. GP must be determined dynamically based on an equation, or system of equations that factor in all three of these variables.

## EPGP Equations

As mentioned in Section III.A, the principal mathematical challenge in EPGP is coming up with a reasonable system of equations for determining GP values in a manner that is proportional to their subjective worth. The equations used in the development of this bot were already well established.

Reference Equation 2 for the equations in determining gear point values. The Gear Points equation is a function of Item Value, which is determined using the system of equations defined by Equation 3. Item value is a function of item-level, a dynamically determined value pulled from an external service, and quality, which is a fixed enumeration that describes an item’s rarity.

The slot value (SV) in the gear points equation is a fixed number that can be determined from reference Table 1.

Equation 2. Gear Points Equation

Equation 3. Item Value Equations

|  |  |
| --- | --- |
| Slot | Slot Value (SV) |
| Shield, Ranged Weapon (non-Hunter), Wrist, Neck, Back, Ring, Offhand, Relic, Idol, Libram, One-handed Weapon (Hunter) | 0.5 |
| Shoulders, Trinket, Feet, Hands, Waist | 0.75 |
| Head, Chest, Legs, Two-handed weapon (Hunter) | 1 |
| Ranged Weapon (Hunter), One-handed Weapon (non-Hunter) | 1.5 |
| Two-handed weapon (non-Hunter) | 2 |

Table 1. Slot Value reference

## The “Hoarding” problem

An issue that many WoW accounting system encounter is a tendency for members to “hoard” their gained capital in order to guarantee that they will be able to acquire specific high-impact rewards when the opportunity arises. This tendency to hoard, while harmless at the individual level, harms the guilds overall ability to adequately progress into harder game content. In order to strengthen the team collective, members must be encouraged to claim as many in-game upgrades as possible, even if those upgrades do not represent the highest possible value.

EP hoarding is addressed by applying a flat decay to all EP and GP values at a fixed interval, generally per week. Because only temporal EP and GP values are accounted, the later a user takes advantage of their PR lead, the less benefit they receive. The more aggressive the decay used, the more individually punishing hoarding becomes.

The chief drawback to decay is that is can be overly punitive to members who miss an EP-gaining event. Since decay progressively diminishes the value of past EP earned, users can fall behind very quickly in systems that use an aggressive decay. Guild administrators must decide carefully how much they wish to punish raid absences when deciding on a decay rate.

## The “New Raider” problem

A common problem with EPGP and similar loot accounting systems that value ratios over raw currency, is that they have a tendency overvalue new users who enter the system. Due to the progressive nature of WoW raid content, guild administrators usually seek to reward veteran members more favorably than new users, so that they can better trust that scarce resources remain active in aiding the guild achieve collective goals.

Consider Table 2, which evaluates a case where two raiders co-exist over a 14-week raid cycle, 50 EP is awarded every week, and EPGP is decayed at a rate of 20% weekly. In week 1, Raider 1 takes an item that is valued at 250 GP. They continue to raid every week, without missing a single event over a 14-week cycle. Week 6 sees the introduction of Raider 2, whom despite being absent for 6 weeks has rocketed ahead of Raider 1 in just a single week of activity. This might allow Raider 2 to claim a high-value item that Raider 1 has been seeking for nearly two months.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | EP1 | GP1 | PR1 | EP2 | GP2 | PR2 |
| Raid 1 | 0 | 251 | 0 |  |  |  |
| Raid 2 | 50 | 200.8 | 0.249004 |  |  |  |
| Raid 3 | 90 | 160.64 | 0.560259 |  |  |  |
| Raid 4 | 122 | 128.512 | 0.949328 |  |  |  |
| Raid 5 | 147.6 | 102.8096 | 1.435664 |  |  |  |
| Raid 6 | 168.08 | 82.24768 | 2.043583 | 1 | 1 | 1 |
| Raid 7 | 184.464 | 65.79814 | 2.803483 | 50.8 | 1 | 50.8 |
| Raid 8 | 197.5712 | 52.63852 | 3.753358 | 90.64 | 1 | 90.64 |
| Raid 9 | 208.057 | 42.11081 | 4.940702 | 122.512 | 1 | 122.512 |
| Raid 10 | 216.4456 | 33.68865 | 6.424881 | 148.0096 | 1 | 148.0096 |
| Raid 11 | 223.1565 | 26.95092 | 8.280105 | 168.4077 | 1 | 168.4077 |
| Raid 12 | 228.5252 | 21.56074 | 10.59914 | 184.7261 | 1 | 184.7261 |
| Raid 13 | 232.8201 | 17.24859 | 13.49792 | 197.7809 | 1 | 197.7809 |
| Raid 14 | 236.2561 | 13.79887 | 17.12141 | 208.2247 | 1 | 208.2247 |

Table 2. Case Study: New Members

This problem is combatted is by setting a GP floor value and an initial GP value, these two values do not need to be the same. Consider Table 3. Which represents the same case study, but with an initial GP value of 50, and a GP floor set to 25. In this case, Raider 2 does not overtake Raider 1 until their fourth consecutive week of raiding in which neither player has claimed a single item.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | EP1 | GP1 | PR1 | EP1 | GP1 | PR2 |
| Raid 1 | 0 | 301 | 0 |  |  |  |
| Raid 2 | 50 | 240.8 | 0.207641 |  |  |  |
| Raid 3 | 90 | 192.64 | 0.467193 |  |  |  |
| Raid 4 | 122 | 154.112 | 0.791632 |  |  |  |
| Raid 5 | 147.6 | 123.2896 | 1.197181 |  |  |  |
| Raid 6 | 168.08 | 98.63168 | 1.704118 | 1 | 50 | 0.02 |
| Raid 7 | 184.464 | 78.90534 | 2.337788 | 50.8 | 40 | 1.27 |
| Raid 8 | 197.5712 | 63.12428 | 3.129877 | 90.64 | 32 | 2.8325 |
| Raid 9 | 208.057 | 50.49942 | 4.119987 | 122.512 | 25.6 | 4.785625 |
| Raid 10 | 216.4456 | 40.39954 | 5.357625 | 148.0096 | 25 | 5.920384 |
| Raid 11 | 223.1565 | 32.31963 | 6.904673 | 168.4077 | 25 | 6.736307 |
| Raid 12 | 228.5252 | 25.8557 | 8.838482 | 184.7261 | 25 | 7.389046 |
| Raid 13 | 232.8201 | 25 | 9.312805 | 197.7809 | 25 | 7.911237 |
| Raid 14 | 236.2561 | 25 | 9.450244 | 208.2247 | 25 | 8.328989 |

Table 3. Case Study: New Members with GP Initialization

# BuzzBot Technical Overview

The BuzzBot application was written in C# using version 3.1 of .NET Core. Integration with Discord is achieved using Discord.Net; an open-source asynchronous API wrapper for Discord. The EPGP module UI and initial milestone features were modeled off the EPGP Discord Bot written by Korkd (Korkd, n.d.). The key functions of the bot are broken into the following modules:

* Raid Module
  + Responsible for managing active raid events, and automated EPGP operations relating to active raids.
* EPGP Module
  + Responsible for all accounting operations related to the EPGP implementation.
* Bank Module
  + Provides ClassicGuildBank integration through their RESTful API.

# Raid Module

The raid module is responsible for managing active raid events, and automating EPGP account operations related to raids in progress.

|  |  |  |  |
| --- | --- | --- | --- |
| **RAID MODULE - !raid command** | | | |
| Command | Example | Admin? | Summary |
| begin {template id} | !raid begin raid1 | Y | Begins a new raid event and prints the signup UI to Discord |
| startnow | !raid startnow | Y | Starts the active raid event immediately |
| extend {time} | !raid extend 60 | Y | Extends the active raid event by the provided duration (minutes) |
| end | !raid end | Y | Ends the active raid event immediately |
| template | !raid template | Y | Prints all configured raid templates |
| kick {user} | !raid kick Jeff | Y | Kicks a user from the active raid event |
| update {templateId} {key} {value} | !raid update raid1 2 20 | Y | Updates a raid template using the provided key-value-pair |
| add {id} {capacity} {startBonus} {endBonus} {timeBonus} {timeBonusIntervalMinutes} {raidDurationMinutes} {signupDurationMinutes} | !raid add raid2 40 15 15 3 15 180 30 | Y | Adds a new raid template using the provided configuration values |
| delete {templateId} | !raid delete raid1 | Y | Deletes a raid template |

## Raid Templates

Raid templates are the cornerstone of the raid module. They allow guild administrators to save raid configurations and quickly initialize new events.

After a fresh install, no raid templates are actively configured.

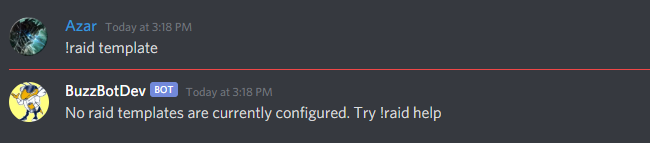


Figure 1. Raid Templating after fresh install

To install a new raid template, an admin user must call *add* command.

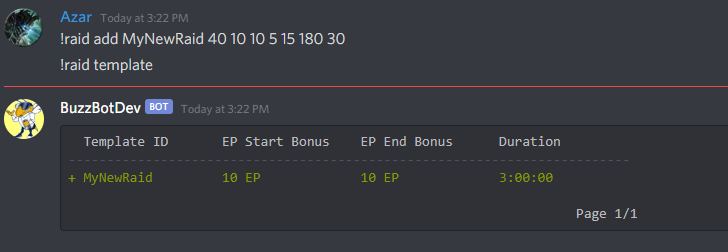


Figure 2. Adding a new raid template

Reference Figure 2. The admin user called *!raid add MyNewRaid 40 10 10 5 15 180 30*. This added a new raid titled MyNewRaid and initialized its configuration. Calling *!raid template* will allow the user to see all templates that have been configured. If the user follows the *!raid template* command with a specific template ID, they will be able to inspect the template configuration.

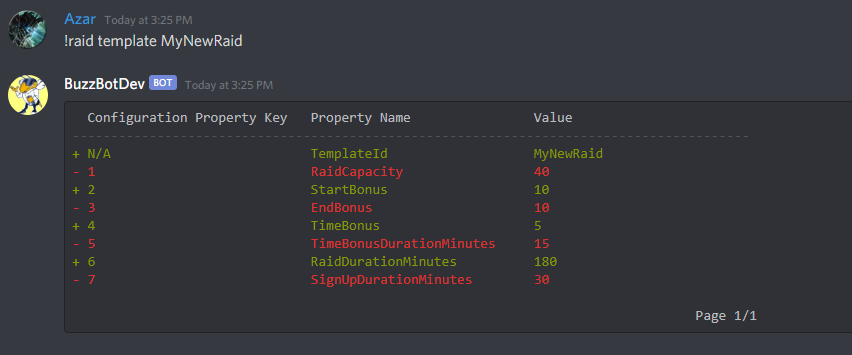
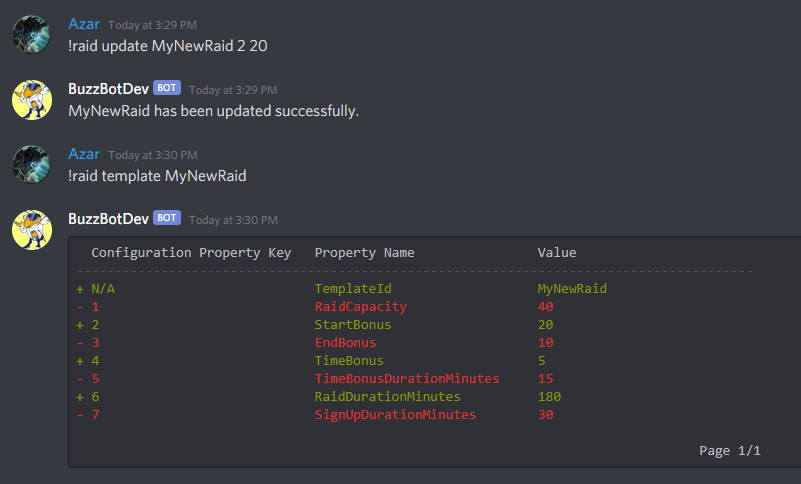


Figure 3. Inspection of Template Configuration

Reference Figure 3. The *!raid template {templateId}* command allows the user to see they key-value-pairs of a template. Any admin user can use these pairs to update a template configuration using the *!raid update* command.



## Raid Events

Once raid templates have been configured. Admin users are able to initialize new raid events using the *!raid begin* command



Figure 4. New Raid Event

|  |  |  |
| --- | --- | --- |
| Key | Item Name |  |
| 1 | Signup tracker | Tracks the number of members who have enrolled in the raid event over the raid’s max capacity |
| 2 | Raid leader | Shows who initialized this raid event |
| 3 | Raid duration | The configured duration for this raid event |
| 4 | EP start bonus | The amount of EP that will be awarded to every enrolled member when the raid begins |
| 5 | EP time bonuses | The amount of EP that be awarded to every enrolled member during the raid, and the interval at which the EP will be awarded |
| 6 | EP end bonus | The amount of EP that be awarded to every enrolled member when the raid ends |
| 7 | Raid roster | The active enrolled roster in the raid |
| 8 | Raid footnotes | Additional notes that have been provided. In this example, the guild has programmed the bot to charge a marginal GP cost to people that get items for “free”. This cost is determined based on market value of a comparable trade good. |
| 9 | Signup mechanism | This is the UI mechanism that discord provides for users to interact with the bot. Users can react to the event using an emote that corresponds to their desired role. |

Table 4. Raid event item definitions

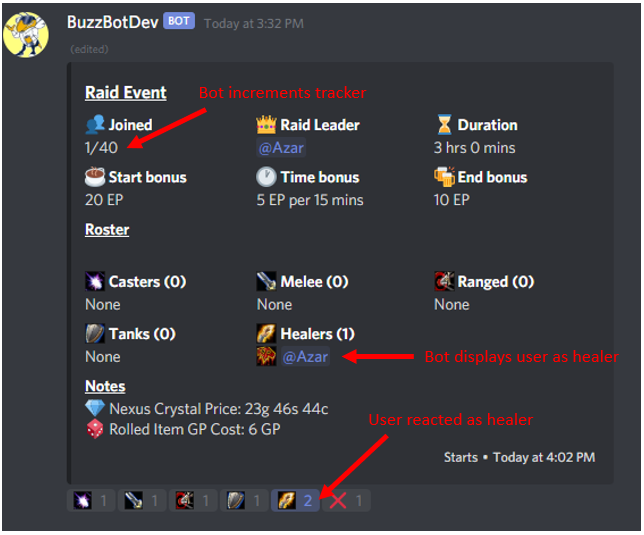


Figure 5. Raid signup

Once a raid event has been initialized with the *!raid begin* command, users will have the entire configured signup duration to react and self-enroll in the event until the designated start time. Once the signup duration has expired, the bot will begin the event and immediately award all enrolled users the “start bonus”.

If the raid admin wishes to start the event early, they may do so by using the *!raid startnow* command.

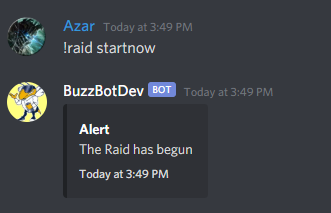


Figure 6. Raid startnow command

After the raid has begun, the bot will manage the active raid event. It will award all enrolled users with the “time bonus” EP at the configured interval until the raid duration has expired. At that point it will award all users with the “end bonus” EP and print a summary of the raid event.

If an admin wishes to end an event before the configured duration, they may do so using the *!raid end* command. Figure 7 demonstrates an ended raid, as well as a known bug where start and end times are not printed correctly for raids that are started or ended early.

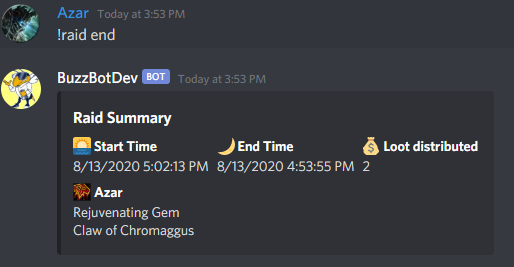


Figure 7. Concluded raid event

# EPGP Module

The EPGP Module is responsible for all accounting and book-keeping for the loot distribution system. Table 5 provides a summary of all EPGP module commands.

|  |  |  |  |
| --- | --- | --- | --- |
| **EPGP MODULE - !epgp command** | | | |
| Command | Example | Admin? | Summary |
| add {user} | !epgp add Jeff | Y | Adds a new discord user to the bot. By convention, the bot will use the nickname of the discord account as the primary alias, and determine the class based on the user role. |
| alias {user} {aliasName} {aliasClass} | !epgp alias Jeff Geoff Hunter | Y | Adds a new alias to a user, also called alternate characters in game. |
| audit {user} | !epgp audit Jeff | Y | Audits all EP and GP transactions for the specified user |
| config | !epgp config | Y | Prints the EPGP configuration settings |
| csv | !epgp csv | Y | Privately messages the calling user a CSV file with all user EPGP data |
| deletealias | !epgp deletealias Geoff | Y | Deletes the specified alias from the system, while retaining the parent user |
| deleteuser | !epgp deleteuser Jeff | Y | Deletes the specified user from the system |
| ep | !epgp ep Geoff 20 | Y | Manually awards EP to the user |
| gp | !epgp gp Geoff 20 !epgp gp Geoff Claw of Chromaggus | Y | Manually awards GP to a user, either based on value or calculated from the provided loot |
| pr | !epgp pr !epgp pr Geoff Jeff |  | Prints the priority order of all users in the system. Optional parameters can filter this list based on role or username. |
| rolls | !epgp rolls Claw of Chromaggus | Y | Awards an item to a user without using default GP calculations. |
| switch | !switch |  | Allows a user to change their active alias |
| undo | !epgp undo {guid} | Y | Allows a user to delete an EPGP transaction. Optional ID parameter, will default to the latest manual transaction. |

Table 5. EPGP Command Summary

## Configuration

The *!epgp config* command will print a summary of the bot configuration, as well as all of the key-value-pairs required to update the configuration. Using *!epgp config {int} {int}* will allow a user to update the configuration using those provided key value pairs.

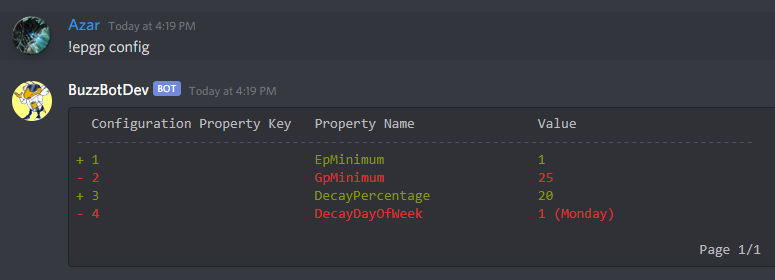


Figure 8. EPGP Configuration

### EP Minimum

The EP Minimum field sets the floor EP value that a user can have. Manual EP modifications and decay will not be able to reduce a user below the specified value.

### GP Minimum

The GP Minimum field sets the floor GP value that user can have. Manual GP modifications and decay will not be able to reduce a user below the specified value. As mentioned in previous sections, this value can be a powerful tool to reduce the impact of new users into the system.

### Decay Percentage

This sets the weekly decay percentage across the entire EPGP system.

### Decay Day of Week

Configuring the Decay Day of Week will set which day the decay gets applied on the system. The values are equal to .NET’s native DayOfWeek enum.

## Adding Users

At a minimum, all Discord participants in the guild server will need a primary alias to participate in raid events. The first user added to the bot will always be added per Discord name and role conventions. Reference Figure 9. When adding the discord user “Azar”, the bot will automatically generate a primary alias and assign the “Shaman” class, based off the role. It is imperative that the name and role be assigned correctly before the user is added to the system.

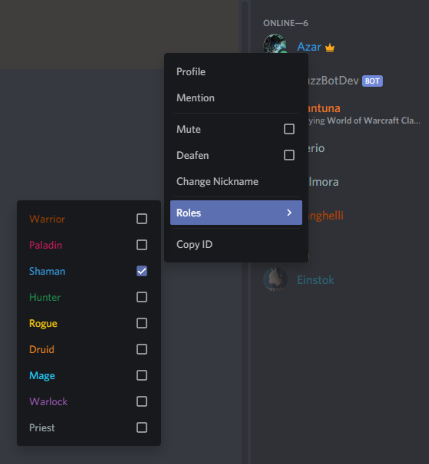


Figure 9. Assigning User Roles

Users will be added automatically the first time they enroll in a raid event, but guild admins are also able to add them manually using the *!epgp add* command.

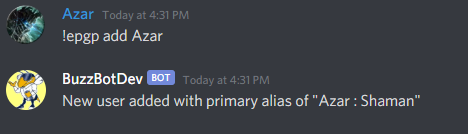


Figure 10. Adding a new user to the Bot

Once a user has been created, additional aliases can be assigned using the *!epgp alias* command. This will allow users to manage multiple game accounts/characters across a single Discord account.

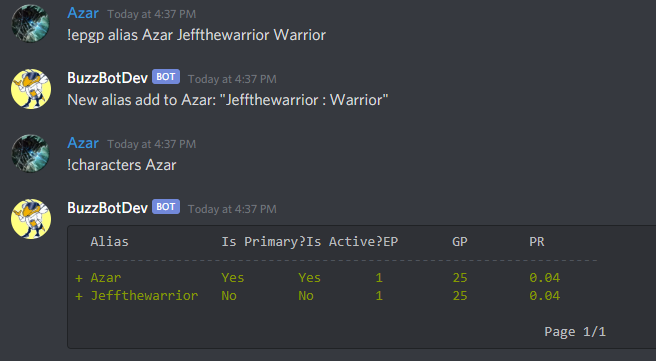


Figure 11. Adding an alias to an existing user.

## Switching Active Aliases

If two or more user aliases have been added, users are able to change their active alias using the *!switch* command. This changes the which alias gets awarded EP and GP during assignments.

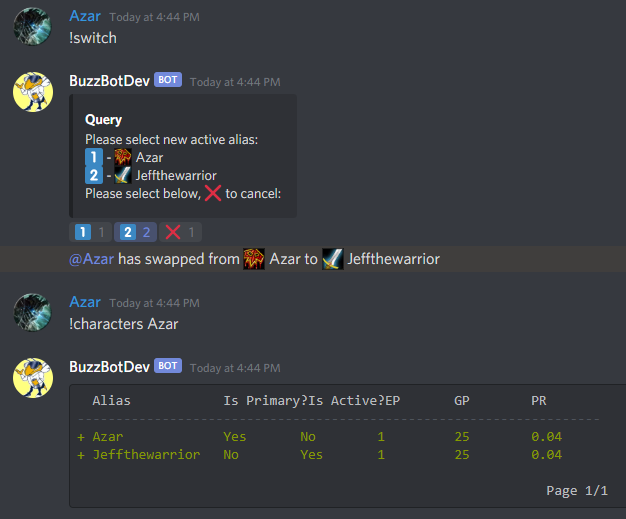


Figure 12. Switching Active Alias

## Awarding EP and GP

Most EP awards will be handled automatically during raid events. If an admin user wishes to make EP awards manually. They can do so using the *!epgp ep* command.

GP awards are generally done using item name conventions. BuzzBot has a robust data store of all items that exist in Classic WoW. If an admin user does a *!epgp gp* assignment using an item query string, the bot will query its database for items that correspond to the query, and generate the GP values based on the formulas described in section III.B. If multiple items correspond simultaneously to the query, the bot will query the user to select from a list.

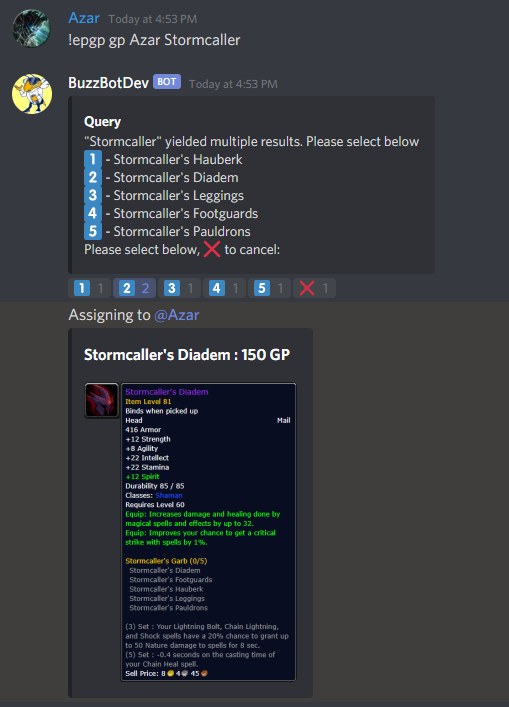


Figure 13. GP assignment from Item

## Determining User Priority

There are three ways that users may query priority lists using the *!epgp pr* command.

* *!epgp pr* without additional parameters will print the entire priority list.
* *!epgp pr {name} {name} {name} etc…* will print a priority list including the specified users
* *!epgp pr {@mention} {@mention} {@mention} etc…* will print a priority list including only the users and roles that have been mentioned.

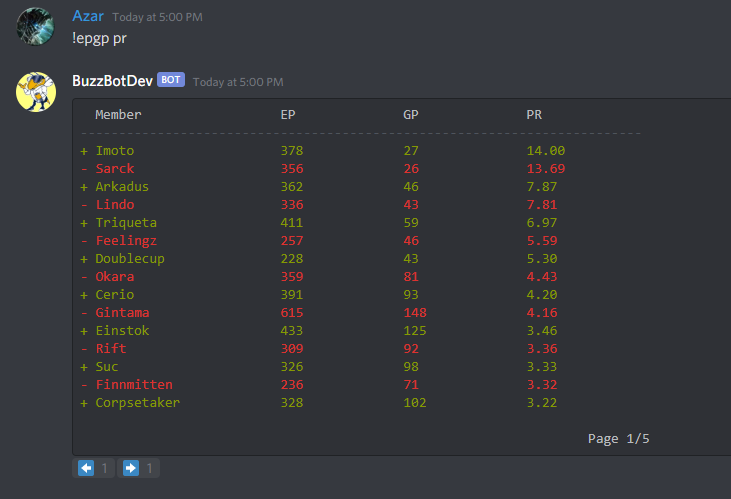


Figure 14. Printing Entire Priority List

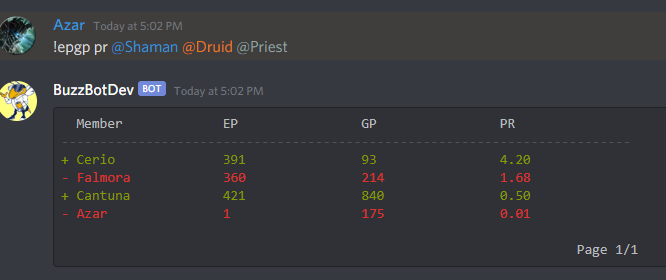


Figure 15. Printing Priority List of Roles

## Auditing Users

The *!epgp audit* command will print all EP and GP transactions that have ever been processed for the provided user. This is a useful tool for maintaining transparency, as well as for clearing up accounting discrepancies.

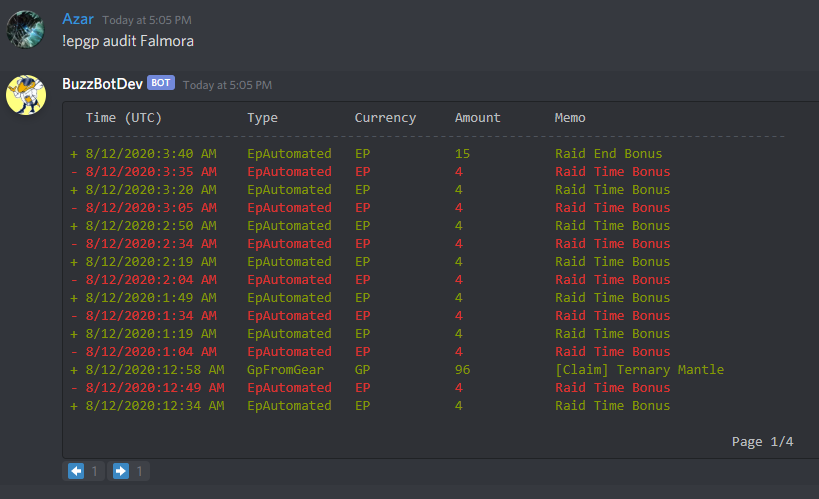


Figure 16. User Auditing

A user’s EP and GP should always sum to the total number of transactions that user has received. Conversely, when a value changes there should always be a transaction to reflect that change. To protect against database tampering, users may use the *!epgp reconcile* command to verify that their account values reflect their transaction history. If the database has been manipulated, this command will reveal the variance.

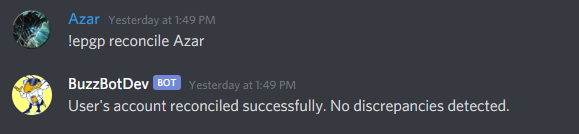


Figure 17. Successfully reconciled user account.

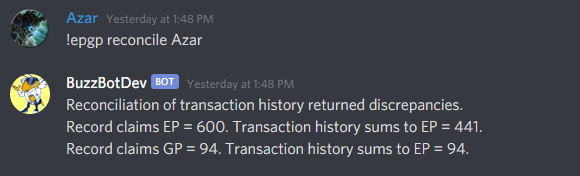


Figure 18. Reconciliation reveals database tampering.

## Removing EPGP Transactions

If an EPGP transaction is entered in error, an admin user can purge it from the record using the *!epgp undo* command. By default, this command will assume the last-entered EPGP transaction and prompt the admin to confirm.

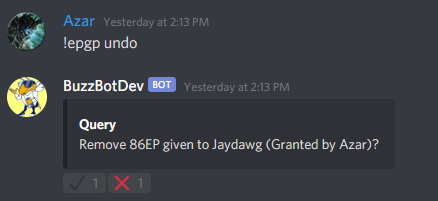


Figure 19. Undo command.

If a user would like to remove a specific EPGP transaction, they must provide the bot the GUID for the transaction. Transaction GUIDs can be determined using an admin-restricted hidden column in the *!epgp audit* print summary.

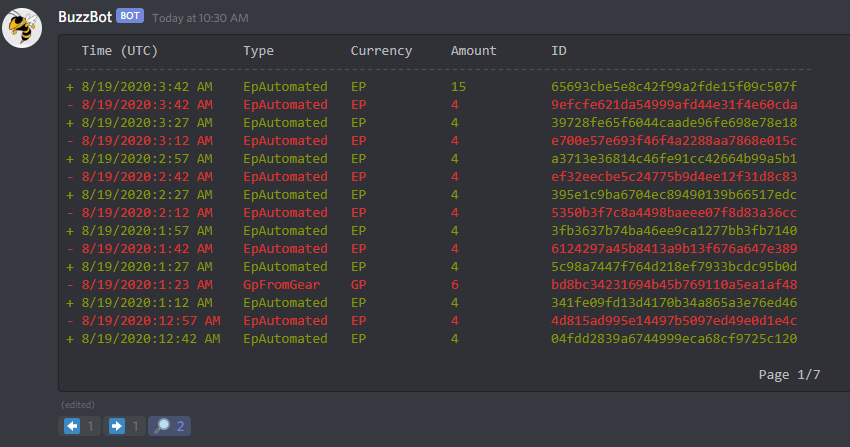


Figure 20. Magnifying glass icon reveals a hidden ID column.

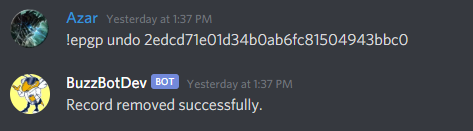


Figure 21. Undo command for targeted EPGP transaction

# Software Overview

The core framework of BuzzBot was built on top of v2.2.0 of Discord.Net. This library was designed to be implemented in a manner that closely resembles .NET Core’s native controller pattern. All user input that the bot detects is routed to a class called the CommandHandlingService. This service is responsible for determining whether the input represents a valid user command, creating a new command scope, and passing the command in its entire context to the CommandService instance for execution.

CommandService determines which module (if any) has been configured to handle the command and uses the scope service provider to resolve that module and all dependent services. These instances survive for the lifetime of the command execution, and then are disposed by the container.

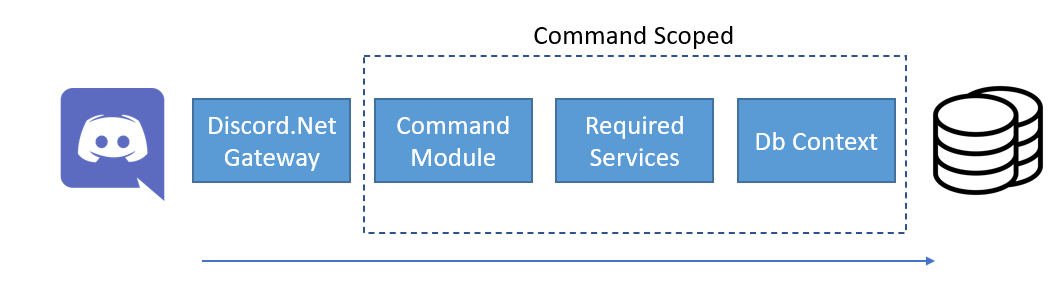


Figure 22. Command handling sequence overview.

## Core Services

### Alias Service

### User Service

### Item Service

### Audit Service

### Raid Service

### EPGP Service

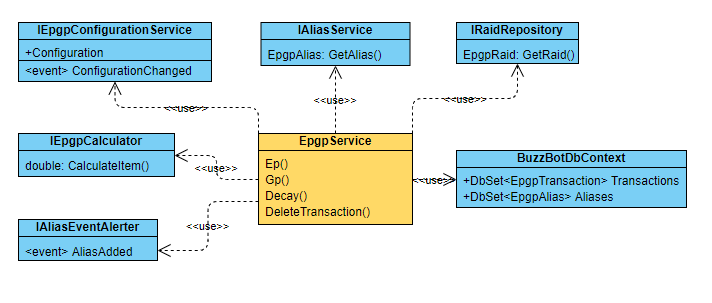


Figure 23. EPGP Service Class Diagram

The purpose of the EPGP Service is to handle all EPGP transactions within the application. The simplest use case of an EPGP transaction is a manual entry.

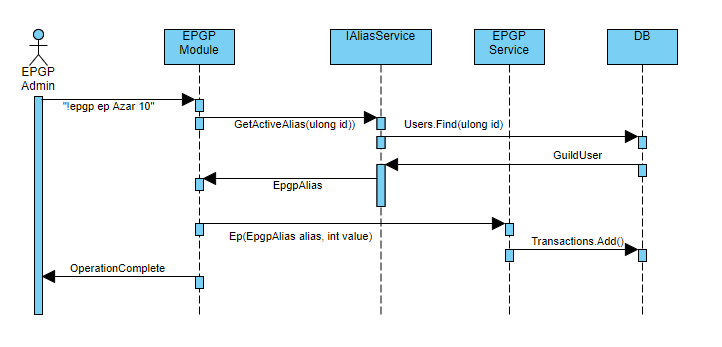


Figure 24. Manual EP/GP entry sequence diagram.

Figure 25 illustrates one of the most complex use cases for the EPGP service. A case where a user wishes to assign GP tied to a specific item, but they provide the bot an ambiguous query string for the item.

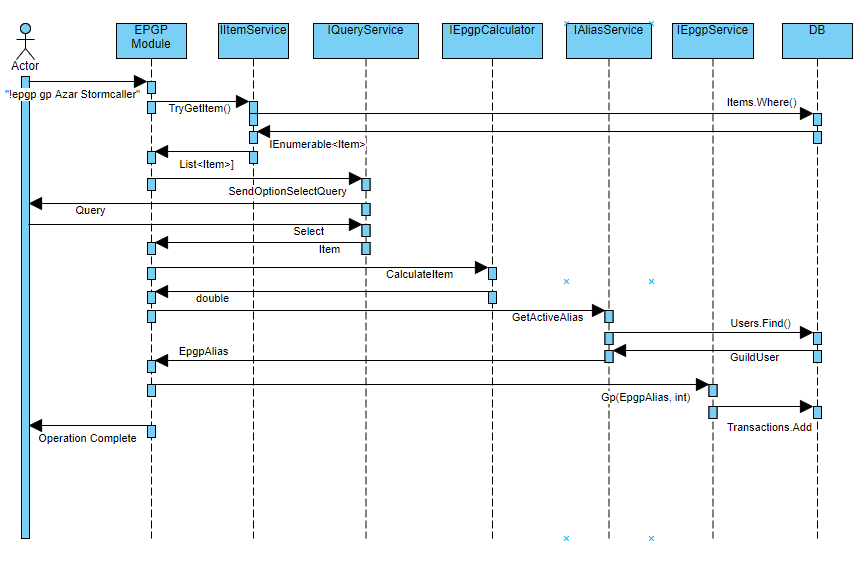


Figure 25. GP Assignment for ambiguous item query

## Utility Services

### Page Service

### Documentation Service

### Emote Service

### Query Service

# Future Development