

SOFTWARE REQUIREMENTS FOR REUNIONLOG

REUNION

L^AT_EX

Lead Programmer and Lead Designer - NICKGISMOKATO

Design contributor - BJ

Design contributor & tester - CASPER

Written by
Nick LAURSEN

Preface

This is the documentation of the requirements we want to follow when creating the program REUNIONLOG. We will be following the SOLID principles. This software is a program meant for the guild REUNION in the game WORLD OF WARCRAFT. This software will use the API from WARCRAFTLOGS.

Most of these requirements have been gathered from multiple months of pre-gathering data from the WARCRAFTLOGS API. This has been done by creating a *proof-of-concept* program with PYTHON.

This software is under the MIT LICENSE. Read LICENSE for more information.

Contents

Contents	2
1 WarcraftLogs and the API	5
1.1 Website	5
1.1.1 Overall form	5
1.1.2 Specific to our needs	5
1.2 Documentation for the API	6
1.2.1 GraphQL	6
1.2.2 Limitation of the API	6
1.3 Integration from the API to C#	7
1.3.1 Authentication	7
1.3.2 Query Strings	8
1.3.3 Data extracted from the API	8
2 Our software main requirements	9
2.1 Authentication	9
2.2 Events	9
2.3 Query Strings	9
2.4 .CSV file	10
3 Flow of ReunionLog	11
3.1 Simple Overview	11
3.2 Needs for each main Requirements	11
4 Responsibilities for our requirements	12
4.1 Responsibilities	12
4.2 UML Diagram	12

A	Figures	13
A.1	Pictures	13
B	Schema	14
B.1	Generic Types	14
B.2	Overview	15
B.3	CharacterData	16
B.3.1	Character	16
B.3.2	Example	20
B.4	GameData	21
B.4.1	GameAbility	25
B.4.2	GameAchievement	25
B.4.3	GameAffix	25
B.4.4	GameClass	25
B.4.5	GameEnchant	25
B.4.6	GameFaction	26
B.4.7	GameItem	26
B.4.8	GameItemSet	26
B.4.9	GameMap	26
B.4.10	GameNPC	26
B.4.11	GameZone	26
B.4.12	Example	26
B.5	GuildData	28
B.5.1	Guild	28
B.5.2	Example	29
B.6	ProgressRaceData	30
B.6.1	Example	30
B.7	RateLimitData	31
B.7.1	Example	31
B.8	ReportData	32
B.8.1	Report	32
B.8.2	Example	36
B.9	UserData	37
B.9.1	User	37
B.9.2	Example	37
B.10	WorldData	38
B.10.1	Encounter	38
B.10.2	Expansion	39
B.10.3	Region	39
B.10.4	Server	39
B.10.5	Subregion	39
B.10.6	Zone	39
B.10.7	Example	40
B.11	Pagination Objects	41
B.11.1	CharacterPagination	41
B.11.2	GameAchievementPagination	41
B.11.3	GameAbilityPagination	41
B.11.4	GameEnchantPagination	41
B.11.5	GameItemPagination	42

B.11.6	GameItemSetPagination	42
B.11.7	GameMapPagination	42
B.11.8	GameNPCPagination	42
B.11.9	GameZonePagination	43
B.11.10	GuildAttendancePagination	43
B.11.11	GuildPagination	43
B.11.12	ReportEventPaginator	43
B.11.13	ReportPagination	43
B.11.14	ServerPagination	44
B.12	Important Types/Enums	45
B.12.1	CharacterRankingMetricType	45
B.12.2	EventDataType	45
B.12.3	ExternalBuffRankFilter	45
B.12.4	FightRankingMetricType	45
B.12.5	GraphDataType	46
B.12.6	GuildRank	46
B.12.7	HardModeLevelRankFilter	46
B.12.8	HostilityType	46
B.12.9	KillType	47
B.12.10	LeaderboardRank	47
B.12.11	RankingCompareType	47
B.12.12	RankingTimeframeType	47
B.12.13	ReportRankingMetricType	47
B.12.14	RoleType	47
B.12.15	SubscriptionStatus	48
B.12.16	TableDataType	48
B.12.17	ViewType	48
B.12.18	_DirectiveLocation	48
B.12.19	_TypeKind	49

Bibliography

50

Chapter 1

WarcraftLogs and the API

Abstract

This chapter will go through WARCRAFTLOGS and the API correlating. For this chapter we will not go through the actual documentation but rather give a short refer to the documentation and lay out the most important aspect from the site and the API

1.1 Website

The website for WARCRAFTLOGS¹ is a popular website used by guilds to gather data to one single site. This is done through multiple addons and their own in-house software.

1.1.1 Overall form

The data that can be collected is both in the form of GUILD data, RAID data, DUNGEON data, CHARACTER data and much more. These can be accessed for all through the website. This can be done by everyone anonymously.

The website uses GraphQL to display most of their data. Both in tables both also in graphs and tables containing graphics. This gives an easier overview for most users. They also have some options you can choose for the specific data you want to be showed.

A downside to this approach is that a lot of the specific options is not showed. One could reason the "simpler" design is because they want all users to use their website, no matter their technical background.

1.1.2 Specific to our needs

What described in **Section 1.1.1** sound really great and reasonable useful for most users case. This is indeed the case for most users, but if you want the information not available on the site or you want another way to represent the data, then the site is not for you. This is why we will be using the data given to us by the API.

The only need we have of the website is to check if the information we get is also the information displayed on the site.

¹<https://www.warcraftlogs.com/>

1.2 Documentation for the API

The API and its documentation is, for a lack of a better word, idiotic made. There exists two API's. Version 1 and version 2. We will be using the latter. This version has "better" documentation than its counterpart and uses OAUTH 2.0 for its API authentication. The documentation can be found two places. For authentication documentation you can find it at Warcraftlogs 2023 and you can find the actual command call documentation at WarcraftLogs and GraphQL 2023.

1.2.1 GraphQL

The first thing we should worry about is the authentication. As mentioned in the **Preface** we have already made successful connection, therefore this will be discussed later.

GraphQL is the schema of how to make calls to the API. This is done by sending you authentication and a "Query" call. This is in simple terms just a string with specific data. This data is both used to tell WARCRAFTLOGS server what you would like to receive about also where in the schema the server would have to lookup this data. Clearly there is a need to create these query strings.

Therefore we will be using GraphQL library from GRAPHQL-DOTNET. More information can be found at GraphQL 2023. If you were to look at WarcraftLogs and GraphQL 2023 you would find there is a lot of commands you can send. Therefore it is important to create an object which can be agile and create all the necessary strings which we will be using.

1.2.2 Limitation of the API

When it comes to the API a lot of limits start to show. Some in the documentation and some in the actual API itself. Let us start talking about the API itself. When on a free tier, i.e not subscribed to WARCRAFTLOGS website, you can at maximum make 3.600 calls to their server pr. hour. This inherently doesn't sound that bad, but you will later see in depths why this is a bad system.

I will agree that this stops most novice DDOS attacks to their server, but since the introduction of the second version of their API, multiple clients can easily be set up and could potentially help in the DDOS attacks. Let us now explain why this rate limit is bad when you want to make a software that pulls a great amount of data.

Let us say you had a guild that was raiding and you both raided MYTHIC and HEROIC during the same log. Let us furthermore assume you want the data for when they die and how many pulls they were on, but only for the mythic. By scouring through their schema documentation (WarcraftLogs and GraphQL 2023), you find the Report object inside the documentation. This is essentially what you want. So you create a query string with report as first argument and then you scour through the documentation and find the event option which deliver a ReportEventPaginator which is just a list event in the log file. This gets returned as data in form of a JSON string. Let us see the documentation for it:

"A set of paginated report events, filterable via arguments like type, source, target, ability, etc. This data is not considered frozen, and it can change without notice. Use at your own risk." (WarcraftLogs and GraphQL 2023)

This object has an argument you can filter from called difficulty. The documentation for this is as follow:

”**difficulty**: Optional. Whether or not to filter the fights to a specific difficulty. By default all fights are included.” (WarcraftLogs and GraphQL 2023)

This is brilliant. We can now filter our `Event` to `mythic` only. Later down the line is argue that `difficulty` is an integer value. Great so now we just need to figure out what integer value. This is the first hurdle we found when trying this. No actual documentation is given for the arguments, except if it is an object or enum type which they have created. So we set up in PYTHON a script to run through with value n where $n \in \{-1000, -999, \dots, 999, 1000\}$. Then we checked if there where any difference between the JSON strings we received and we got that there was no difference for any n value.

Clearly any user would be confused. Why would an API give you an option and then not document said option and actually do nothing with this option. Your first instinct would now be to contact WARCRAFTLOGS. So you write them an email and there response they send you back can be seen in fig. A.1. So you go to their discord find the thread about the API and ask you question. The intention of this documentation is not to out anyone from their support team and therefore I will not share the conversion I had. But trust us when we say that if was probably the most useless information and support we have ever received.

So how did we fix it, you may ask? We looked at the documentation and again read what was written under the `difficulty` argument. The `difficulty` argument says it filter fights. So we then took a look at the `ReportFight` object. We tested this and sure enough it worked. So to simplify what we now have to do if we wish to get the event data only for MYTHIC. First we need to get all the fight data that had that difficulty i.e the `fightIDs`. This is done with the `ReportFlight` object. Then we have to sort our `ReportEventPaginator` to the `fightIDs`. Now we have used two calls of our 3.600 rate limit.

In our demo in PYTHON it became clear that we also had to pull a lot more data than just fight i.e pulls and so on. We now have to pull every single fiights, every single name from that fight and the event itself. This gives us a rough estimate:

$$l \cdot (a_{\text{Fights}} + a_{\text{Event}} + a_{\text{name}} \cdot p),$$

where p is the amount of players in any fights and l is the amount of different logs we want to pull data from. Since we cannot get `fightIDs` whilst also getting the event to only get data from the mythics, we would need to do this over two separate calls and also we cannot pull names just from the event log the way we want it, so we have to do it separately.

This is probably the most irritating limit of the API and therefore we would need to make sure our software can handle this limit in case our request limit get met.

1.3 Integration from the API to C#

As discussed in 1.2 it is clear that we would need many objects to handle everything involving WARCRAFTLOGS API and it’s authentication. Clearly we would need objects handling our query strings, *all of them*, handling of the authentication and most importantly handling the data we receive after we have made a call, i.e the response.

1.3.1 Authentication

Creating an authentication for our software is actually quite easy. We don’t need any libraries for the creation of the code or the requester. The only thing we need is an users `client_id` and `client_secret`. The `tokenURL` is a constant i.e

"<https://www.warcraftlogs.com/oauth/token>". Everything else is user input. This is also important but will be dealt with differently.

We will need the user to create the `client_id` and `client_secret` which we will save as local data for further use. This will be done at first time use of the software.

1.3.2 Query Strings

When dealing with query strings within this API, you are in fact dealing with a `json` call within the `GraphQL` schema. As noted in **Section 1.2.1 GraphQL**, we will be using the `GraphQL` library from `Graphql-dotnet` (GraphQL 2023). This library is also under the MIT license, so no issue in license agreements here. It is important to note that FACEBOOK created `GraphQL`. This library is just an implementation to `.NET`. You can easily install it by the following command:

```
$ dotnet add package GraphQL
```

This is the only package needed for this library. We don't need serialization or document caching since we will implement it ourself if needed.

1.3.3 Data extracted from the API

After we have made a call, you would think we are done. Clearly this isn't the case. The `JSON` is what our call return. This data is not just a "pure" `JSON` string. It has layers within its own index. Therefore we need to go through these index layers if we wish to get to the actual data we need. This could either be implemented in the call function or in it's own class.

We will implement this within its own class. The main reason is we don't want to give the call object that responsibility. That lies solely within the respected event objects.

Chapter 2

Our software main requirements

Abstract

In this chapter we will go through all our requirements for our software. This will be done in a list form. This chapter will not be dealing with the responsibilities within the objects.

2.1 Authentication

The authentication part of the software has the following requirements:

- Create an authentication call
- Use the `client_id` and `client_secret` for the authentication call
- Create some `.credential.JSON` file to hold the credentials we get from authentication call
- If multiple calls is made to the API, it should be able to use the `.credential.JSON`
- Delete `.credential.JSON` when program is stopped.

Further more, it should have/use the following:

- Be a static class, maybe with some other type of classes (depends of the rest of the design).
- Check if `client_id` and `client_secret` is given. If not then it should prompt a creation page
- Hold a boolean value `createdCredentials` and it should use it to check whether or not the `.credential.JSON` has been created or exists
- Have a custom exception if authentication failed and should print the exception
- If an error was returned from the API it should also handle the error with an exception.

2.2 Events

2.3 Query Strings

The query string part has the following requirements:

- Create a `JSON String` type to cast to the event call of the software
- Handle different Master type objects from the API
 - This should be handled with a switch and using an enum object.
 - This should handle any error with an exception if wrong or non existing arguments is given.
- It will never hold the `JSON String` type. Only create and then return it

Further more, it should have/use the following:

- Use enum's to handle different API types
 - The enum's should be contained in the same namespace as the query string or should have it own
 - The enum's value holder should not be used
 - The default value of the enum's should return either null or some other empty value types
- It should be as agile as possible i.e it should follow the O principle in SOLID
- If it need to handle mutation of strings or value not a query, it should not handle it but instead call a function that mutate it
- It has to throw exceptions if either
 - Important query values is missing
 - default value from enums is received
 - Any errors are thrown
- No default query string should be returned if any error or exception is detected. It should instead return null.

2.4 .CSV file

The most important part of the .csv file is not that we can use the software you change the design but instead that it is consistent with what design is used if multiple (*or single*) query's are used.

It is important that this software is meant for the REUNION guild and not for all guilds that want to use this software. That being said, we would still make the design part of the .csv handler/creator easy to redo and redesign on. Therefore we would have:

- The creation of the .csv files should be handled by a static object called `CreateCSVFile`
- This object should not hold any data, but instead be given the necessary data from the event object after it has received the data.
- It shouldn't have the need to manipulate the data but instead send the data of to me mutated into the necessary data needed to create the .csv data

Note: This is why we use a static class since we should be able to create multiple .csv files in one run or multiple runs where no exist of the software has happen. This is also more memory safe for the program

Note: Best practice would dictate us to create an `EventHandler` to handle these multiple calls but since this is a static class living in the `Main` class of our program, no real usage or performance gain would come of such practices

- If it encounter an error for any reason, no .csv files should be created. Instead a .log file should be created and show where in the process it was interrupted

Note: This log should be created every time but should be deleted if no exception was thrown or no error was detected

Note: A debug option should exists in the software to determine whether or not the .log file should be deleted after successfully handling a creation call

Note: If option to not delete the .log files, then it need to call the different files some different from each other. The reason is we don't want to overwrite previously written .log files and we want a way to distinguished the different files.

Chapter 3

Flow of ReunionLog

Abstract

In this section we will go through the simple flow of how the software will behave. This is both in terms of how the software access the API and how many times it does but also how calls for object is being handled. Clearly there is a lot to talk about so let us start with an simple overview before diving into the abstract parts of the software.

Clearly, this section in not meant as a substitution for an actual UML diagram and responsibilities table. It is merely meant to give the reader a better understanding of the software before diving into it. Since no actual standard is made for this section, I will try to do my best to come around the software.

3.1 Simple Overview

3.2 Needs for each main Requirements

Chapter 4

Responsibilities for our requirements

Abstract

4.1 Responsibilities

4.2 UML Diagram

Appendix A

Figures

A.1 Pictures

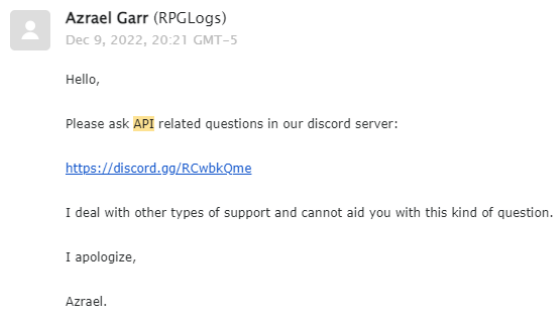


Figure A.1: Mail received from the support team at WARCRAFTLOGS

Appendix B

Schema

B.1 Generic Types

Int:

An integer is a whole number $n \in A \subset \mathbb{Z}$ where A is finite. GraphQL can handle both 32 and 64 bit int, since GraphQL V16. Int is still just a 32 bit integer even though it could support it. Note that the type Int is a signed 32 bit integer.

note: There is no reason to use 64 bit integers since neither WoW or FF14 uses this.

String:

A String is a scalar of a list of characters i.e "This Is a string example". Since String is a scalar, there is theoretically no limits for the length in GraphQL. This hasn't been tested to check whether or not WARCRAFTLOGSAPI support very long strings. Practically there is no need for very long strings.

Boolean:

a Boolean is either true or false. No further explanation is needed.

Float:

A Float is a signed double precision floating point value, i.e float64 or IEEE 754 double-precision binary floating-point format (Wikipedia 2022).

Generic.Enum: A Generic.Enum type doesn't exist within GraphQL. It's up to the developer using GraphQL to implement such types. These can be seen in appendix B.12. They don't need string to around them when writing i.e `dataType: Deaths` where Deaths is of a type `EventData`

Generic.Pagination:

A Pagination object is an object where some of its data types within the object is returned as a list.

Example: Instead of calling three different players within a guild where these players have type `Character` we can just call to the type `CharacterPagination` where you give the arguments `guildID: Int`, `limit: Int`, `page: Int`. From here we can get all members of the guild and then from our end remove all unwanted players. This is smart, since we don't need to make three calls but just one call.

B.2 Overview

Main Call: query

- characterData: [CharacterData](#)
- gameData: [GameData](#)
- guildData: [GuildData](#)
- progressRaceData: [ProgressRaceData](#)
- rateLimitData: [RateLimitData](#)
- reportData: [ReportData](#)
- userData: [UserData](#)
- worldData: [WorldData](#)

Details:

query is the main call and is has type [Query](#). The outputs of this [Query](#) can by any of the Data types.

Example

```
1 query{
2   characterData{
3     //Insert CharacterData calls here
4   }
5   reportData{
6     //Insert ReportData calls here
7   }
8 }
```

B.3 CharacterData

Type CharacterData

- character(id: Int, serverSlug: String, serverRegion: String): Character
- characters(guildID: Int, limit: Int, page: Int): CharacterPagination

Details:

character:

Input List		
Input	Type	Requirement
id	Int	Optional
name	String	Optional
serverSlug	String	Optional
serverRegion	String	Optional
Output Type: Character		

characters:

Input List		
Input	Type	Requirement
guildID	Int	Required
limit	Int	Optional
page	Int	Optional
Output Type: CharacterPagination		

B.3.1 Character

Type Character

- canonicalID: Int!
- claimed: Boolean
- classID: Int!
- encounterRankings(byBracket: Boolean, className: String, compare: RankingCompareType, difficulty: Int, encounterID: Int, includeCombatantInfo: Boolean, includePrivateLogs: Boolean, metric: CharacterRankingMetricType, partition: Int, role: RoleType, size: Int, specName: String, timeframe: RankingTimeframeType): JSON
- faction: GameFaction!
- gameData(specID: Int, forceUpdate: Boolean): JSON


```
- guildRank: Int!
- guilds: [Guild]
- hidden: Boolean!
- id: Int!
- level: Int!
- name: String
- recentReports(limit: Int, page: Int): ReportPagination
- server: Server!
- zoneRankings(byBracket: Boolean,
  className: String,
  compare: RankingCompareType,
  difficulty: Int,
  includePrivateLogs: Boolean,
  metric: CharacterRankingMetricType,
  partition: Int,
  role: RoleType,
  size: Int,
  specName: String,
  timeframe: RankingTimeframeType,
  zoneID: Int): JSON
```

Details:**canonicalID:**

Output Type: Int!

claimed:

Output Type: Boolean

classID:

Output Type: Int!

encounterRankings:

Input List		
Input	Type	Requirement
byBracket	Boolean	Optional
className	String	Optional
compare	RankingCompareType	Optional
difficulty	Int	Optional
encounterID	Int	Required
includeCombatantInfo	Boolean	Optional
includePrivateLogs	Boolean	Optional
metric	CharacterRankingMetricType	Optional
partition	Int	Optional
role	RoleType	Optional
size	Int	Optional
specName	String	Optional
timeframe	RankingTimeframeType	Optional
Output Type: JSON		

faction:

Output Type: GameFaction!

gameData:

Output Type: JSON

guildRank:

Output Type: Int!

guilds:

Output Type: [Guild]

hidden:

Output Type: Boolean!

id:

Output Type: Int!

level:

Output Type: Int!

name:

Output Type: String

recentReports:

Input List		
Input	Type	Requirement
Limit	Int	Optional
page	Int	Optional
Output Type: ReportPagination		

server:

Output Type: Server!

zoneRankings:

Input List		
Input	Type	Requirement
byBracket	Boolean	Optional
className	String	Optional
compare	RankingCompareType	Optional
difficulty	Int	Optional
includePrivateLogs	Boolean	Optional
metric	CharacterRankingMetricType	Optional
partition	Int	Optional
role	RoleType	Optional
size	Int	Optional
specName	String	Optional
timeframe	RankingTimeframeType	Optional
zoneID	Int	Optional
Output Type: JSON		

B.3.2 Example

```
1 query{
2   characterData{
3     character(id: 0000, name: "test"){
4       claimed
5       classID
6       hidden
7       id
8       server{
9         id
10        name
11      }
12    }
13  }
14 }
```

This query call makes the toplevel call to query. Then after that it chooses the characterData call as it sub call category. Thereafter we then further call the character sub call category and gives the required arguments. Then we call to get the data claimed, classID, hidden, id, server where server return id and name as it data.

B.4 GameData

Type GameData

- abilities(limit: Int, page: Int): GameAbilityPagination
- ability(id: Int): GameAbility
- achievement(id: Int): GameAchievement
- achievements(limit: Int, page: Int): GameAchievementPagination
- affix(id: Int): GameAffix
- affixes: [GameAffix]
- class(id: Int, faction_id: Int, zone_id: Int): GameClass
- classes(faction_id: Int, zone_id: Int): [GameClass]
- enchant(id: Int): GameEnchant
- enchants(limit: Int, page: Int): GameEnchantPagination
- factions: [GameFaction]
- item(id: Int): GameItem
- item_set(id: Int): GameItemSet
- item_sets(limit: Int, page: Int): GameItemSetPagination
- items(limit: Int, page: Int): GameItemPagination
- map(id: Int): GameMap
- maps(limit: Int, page: Int): GameMapPagination
- npc(id: Int): GameNPC
- npcs(limit: Int, page: Int): GameNPCPagination
- zone(id: Int): GameZone
- zones(limit: Int, page: Int): GameZonePagination

Details:

abilities:

Input List		
Input	Type	Requirement
limit	Int	Optional
page	Int	Optional
Output Type: GameAbilityPagination		

ability:

Input List		
Input	Type	Requirement
id	Int	Required
Output Type: GameAbility		

achievement:

Input List		
Input	Type	Requirement
id	Int	Required
Output Type: GameAchievement		

achievements:

Input List		
Input	Type	Requirement
int	Int	Optional
page	Int	Optional
Output Type: GameAchievementPagination		

affix:

Input List		
Input	Type	Requirement
id	Int	Required
Output Type: GameAffix		

affixes:

Output Type: [GameAffix]

class:

Input List		
Input	Type	Requirement
id	Int	Required
faction_id	Int	Optional
zone_id	Int	Optional
Output Type: GameClass		

classes:

Input List		
Input	Type	Requirement
faction_id	Int	Optional
zone_id	Int	Optional
Output Type: [GameClass]		

enchant:

Input List		
Input	Type	Requirement
id	Int	Required
Output Type: GameEnchant		

enchants:

Input List		
Input	Type	Requirement
limit	Int	Optional
page	Int	Optional
Output Type: GameEnchantPagination		

factions:

Output Type: [GameFaction]

item:

Input List		
Input	Type	Requirement
id	Int	Requirement
Output Type: GameItem		

item_set:

Input List		
Input	Type	Requirement
id	Int	Required
Output Type: GameItemSet		

item_sets:

Input List		
Input	Type	Requirement
limit	Int	Optional
page	Int	Optional
Output Type: GameItemSetPagination		

items:

Input List		
Input	Type	Requirement
limit	Int	Optional
page	Int	Optional
Output Type: GameItemPagination		

map:

Input List		
Input	Type	Requirement
id	Int	Required
Output Type: GameMap		

maps:

Input List		
Input	Type	Requirement
limit	Int	Optional
page	Int	Optional
Output Type: GameMapPagination		

npc:

Input List		
Input	Type	Requirement
id	Int	Required
Output Type: GameNPC		

npcs:

Input List		
Input	Type	Requirement
limit	Int	Optional
page	Int	Optional
Output Type: GameNPCPagination		

zone:

Input List		
Input	Type	Requirement
id	Int	Required
Output Type: GameZone		

zones:

Input List		
Input	Type	Requirement
limit	Int	Optional
page	Int	Optional
Output Type: GameZonePagination		

B.4.1 GameAbility

Type GameAbility

- id: Int!
- icon: String
- name: String

B.4.2 GameAchievement

Type GameAchievement

- id: Int!
- icon: String
- name: String

B.4.3 GameAffix

Type GameAffix

- id: Int!
- icon: String
- name: String

B.4.4 GameClass

Type GameClass

- id: Int!
- name: String!
- slug: String!
- specs: GameSpec

B.4.4.1 GameSpec

Type GameSpec

- id: Int!
- class: GameClass
- name: String!
- slug: String!

B.4.5 GameEnchant

Type GameEnchant

- id: Int!
- name: String

B.4.6 GameFaction

Type GameFaction

- id: Int!
- name: String!

B.4.7 GameItem

Type GameItem

- id: Int!
- icon: String
- name: String

B.4.8 GameItemSet

Type GameItemSet

- id: Int!
- name: String

B.4.9 GameMap

Type GameMap

- id: Int!
- name: String

B.4.10 GameNPC

Type GameNPC

- id: Int!
- name: String

B.4.11 GameZone

Type GameZone

- id: Float!
- name: String

B.4.12 Example

```
1 query{
2   gameData{
3     ability(id: 420){
4       name
5     }
6     factions{
7       name
8     }
9     class(id: 1, faction_id: 0){
10      name
11      specs{
12        id
13        name
14        slug
15      }
16    }
17  }
18 }
```

This could be a potential query someone would make. Here we get the name of the ability with `id: 420`, all factions names and the name and specs of the class with `id:1` and `faction_id:0`.

B.5 GuildData

Type GuildData

- guild(id: Int, name: String, serverSlug: String, serverRegion: String): Guild
- guilds(limit: Int, page: Int, serverID: Int, serverSlug: String, serverRegion: String): GuildPagination

B.5.1 Guild

Type Guild

- attendance(guildTagId: Int, limit: Int, page: Int, zoneID: Int): GuildAttendancePagination!
- competitionMode: Boolean!
- description: String!
- faction: GameFaction!
- id: Int!
- name: String!
- server: Server!
- stealthMode: Boolean!
- tags: [GuildTag]
- members(limit: Int, page: Int): CharacterPagination!
- currentUserRank: GuildRank
- zoneRank(zoneID: Int): GuildZoneRankings!

B.5.1.1 GuildTag

Type GuildTag

- id: Int!
- guild: Guild!
- name: String!

B.5.1.2 GuildZoneRankings

Type GuildZoneRankings

- progress(size: Int): WorldRegionServerRankPositions
- speed(size: Int, difficulty: Int): WorldRegionServerRankPositions
- completeRaidSpeed(size: Int, difficulty: Int): WorldRegionServerRankPositions

B.5.1.3 GuildAttendance

Type GuildAttendance

- code: String!
- players: [PlayerAttendance]

- startTime: `Float`
- zone: `Zone`

B.5.1.4 PlayerAttendance

Type PlayerAttendance

- name: `String`
- type: `String`
- presence: `Int`

B.5.2 Example

B.6 ProgressRaceData

Type ProgressRaceData

- progressRace(serverRegion: *String*, ServerSubregion: *String*, serverSlug: *String*, zoneID: *Int*, competitionID: *Int*, difficulty: *Int*, size: *Int*, guildID: *Int*, guildName: *String*): *JSON*
- detailedComposition(competitionID: *Int*, guildID: *Int*, guildName: *String*, serverSlug: *String*, serverRegion: *String*, encounterID: *Int*, difficulty: *Int*, size: *Int*): *JSON*

Details:

progressRace:

Input List		
Input	Type	Requirement
serverRegion	String	Optional
serverSubregion	String	Optional
serverSlug	String	Optional
zoneId	Int	Optional
competitionID	Int	Optional
difficulty	Int	Optional
size	Int	Optional
guildID	Int	Optional
guildName	String	Optional
Output Type: JSON		

detailedComposition:

Input List		
Input	Type	Requirement
competitionID	Int	Optional
guildID	Int	Optional
guildName	String	Optional
serverSlug	String	Optional
serverRegion	String	Optional
encounterID	Int	Optional
difficulty	Int	Optional
size	Int	Optional
Output Type: JSON		

B.6.1 Example

B.7 RateLimitData

Type RateLimitData

- limitPerHour: Int!
- pointSpentThisHour: Float!
- pointsResetIn: Int!

B.7.1 Example

```
1 query{  
2   RateLimitData{  
3     limitPrHour  
4     pointSpentThisHour  
5     pointsResetIn  
6   }  
7 }
```

This is a full example of what a query for RateLimitData looks like.

B.8 ReportData

- reportData
 - report (code: `String`): `Report`
 - reports (endTime: `Float`,
guildID: `Int`,
guildName: `String`,
guildServerSlug: `String`,
guildServerRegion: `String`,
guildTagID: `Int`,
userID: `Int`,
limit: `Int`,
page: `Int`,
startTime: `Float`,
zoneID: `Int`,
gameZoneID: `Int`): `ReportPagination`

B.8.1 Report

- report
 - code: `String!`
 - endTime: `Float!`
 - events (abilityID: `Float`,
dataType: `EventDataType`,
death: `Int`,
difficulty: `Int`,
encounterID: `Int`,
endTime: `Float`,
fightIDs: [`Int`],
filterExpression: `String`,
hostilityType: `HostilityType`,
includeResources: `Boolean`,
killType: `KillType`,
limit: `Int`,
sourceAurasAbsent: `String`,
sourceAurasPresent: `String`,
sourceClass: `Int`,
sourceID: `Int`,
sourceInstanceID: `Int`,
startTime: `Float`,
targetAurasAbsent: `String`,
targetAurasPresent: `String`,
targetClass: `String`,
targetID: `Int`,
targetInstanceID: `Int`,
translate: `Boolean`,
useAbilityIDs: `Boolean`,
useActorIDs: `Boolean`,
viewOptions: `Int`,
wipeCutoff: `Int`): `ReportEventPaginator`


```
- exportedSegments: Int!
- fights(difficulty: Int,
  encounterID: Int,
  fightIDs: [Int],
  killType: KillType,
  Translate: Boolean): [ReportFight]
- graph(abilityID: Float,
  dataType: GraphDataType,
  death: Int,
  difficulty: Int,
  encounterID: Int,
  endTime: Float,
  fightIDs: [Int],
  filterExpression: String,
  hostilityType: HostilityType,
  killType: KillType,
  sourceAurasAbsent: String,
  sourceAurasPresent: String,
  sourceClass: Int,
  sourceID: Int,
  sourceInstanceID: Int,
  startTime: Float,
  targetAurasAbsent: String,
  targetAurasPresent: String,
  targetClass: String,
  targetID: Int,
  targetInstanceID: Int,
  translate: Boolean,
  viewOptions: Int,
  viewBy: ViewType,
  wipeCutoff: Int): JSON
- guild: Guild
- guildTag: GuildTag
- owner: User
- masterData(translate: Boolean): ReportMasterData
- playerDetails(difficulty: Int,
  encounterID: Int,
  endTime: Float,
  fightIDs: [Int],
  killType: KillType,
  startTime: Float,
  translate: Boolean): JSON
- rankedCharacters: [Character]
- rankings(compare: RankingCompareType,
  difficulty: Int,
  encounterID: Int,
  fightIDs: [Int],
```

```

    playerMetric: ReportRankingMetricType,
    timeframe: RankingTimeframeType): JSON
- region: Region
- revision: Int!
- segments: Int!
- startTime: Float!
- table(abilityID: Float,
  dataType: TableDataType,
  death: Int,
  difficulty: Int,
  encounterID: Int,
  endTime: Float,
  fightIDs: [Int],
  filterExpression: String,
  hostilityType: HostilityType,
  killType: KillType,
  sourceAurasAbsent: String,
  sourceAurasPresent: String,
  sourceClass: String,
  sourceID: Int,
  sourceInstanceID: Int,
  startTime: Float,
  targetAurasAbsent: String,
  targetAurasPresent: String,
  targetClass: String,
  targetID: Int,
  targetInstanceID: Int,
  translate: Boolean,
  viewOptions: Int,
  viewBy: ViewType,
  wipeCutoff: Int): JSON
- title: String!
- visibility: String!
- zone: Zone
- archiveStatus: ReportArchiveStatus

```

B.8.1.1 ReportFight

- fights
 - averageItemLevel: Float
 - bossPercentage: Float
 - boundingBox: ReportMapBoundingBox
 - classicSeasonID: Int
 - completeRaid: Boolean!
 - difficulty: Int
 - dungeonPulls: [ReportDungeonPull]

- encounterID: Int!
- endTime: Float!
- enemyNPCs: [ReportFightNPC]
- enemyPets: [ReportFightNPC]
- enemyPlayers: [Int]
- fightpercentage: Float
- friendlyNPCs: [ReportFightNPC]
- friendlyPets: [ReportFightNPC]
- friendlyPlayers: [Int]
- gameZone: GameZone
- hardModeLevel: Int
- id: Int!
- inProgress: Boolean
- keystoneAffixed: [Int]
- keystoneBonus: Int
- keystoneLevel: Int
- keystoneTime: Int
- kill: Boolean
- lastPhase: Int
- lastPhaseIsIntermission: Boolean
- layer: Int
- maps: [ReportMap]
- name: String!
- rating: Int
- size: Int
- startTime: Float!
- talentImportCode(actorID: Int!): String
- wipeCalledTime: Float

B.8.1.2 ReportMasterData

- masterData
 - logVersion: Int!
 - gameVersion: Int
 - lang: String
 - abilities: [ReportAbility]
 - actors(type: String,
subType: String): [ReportActor]

B.8.1.3 ReportAbility

- abilities
 - gameID: `Float`
 - icon: `String`
 - name: `String`
 - type: `String`

B.8.1.4 ReportActor

- actors
 - gameID: `Float`
 - icon: `String`
 - id: `Int`
 - name: `String`
 - petOwner: `Int`
 - server: `String`
 - subType: `String`
 - type: `String`

B.8.1.5 ReportArchiveStatus**B.8.2 Example**

B.9 UserData

Type UserData

- user(id: Int): User
- currentUser: User

B.9.1 User

Type User

- id: Int!
- name: String!
- guilds: [Guild]
- characters: [Character]
- battleTag: String

B.9.2 Example

B.10 WorldData

- worldData
 - encounter(id: Int): Encounter
 - expansion(id: int): Expansion
 - expansions: [Expansion]
 - region(id: Int): Region
 - region: [Region]
 - server(id: Int, region: String, slug: String): Server
 - subregion(id: Int): Subregion
 - zone(id: Int): Zone
 - zones(expansion_id: Int): [Zone]

B.10.1 Encounter

Type Encounter

- id: Int!
- name: String!
- characterRankings(bracket: Int, difficulty: Int, filter: String, page: Int, partition: Int, serverRegion: String, serverSlug: String, size: Int, leaderboard: LeaderboardRank, hardModeLevel: HardModeLevelRankFilter, metric: CharacterRankingMetricType, includeCombatInfo: Boolean, className: String, specName: String, externalBufs: ExternalBuffRankFilter, covenantID: Int, soulbindID: Int): JSON
- fightRankings(bracket: Int, difficulty: Int, filter: String, page: String, partition: Int, serverRegion: String, serverSlug: String, size: Int, leaderboard: LeaderboardRank, hardModeLevel: HardModeLevelRankFilter, metric: FightRankingMetricType): JSON

- zone: `Zone!`
- journalID: `Int!`

B.10.2 Expansion

Type Expansion

- id: `Int!`
- name: `String!`
- zones: `[Zone]`

B.10.3 Region

Type Region

- id: `Int!`
- compactName: `String!`
- name: `String!`
- slug: `String!`
- subregions: `[Subregion]`
- servers(limit: `Int`, page: `Int`): `ServerPagination`

B.10.4 Server

Type Server

- id: `Int!`
- name: `String!`
- normalizedName: `String!`
- slug: `String!`
- region: `Region!`
- subregion: `Subregion!`
- guilds(limit: `Int`, page: `Int`): `GuildPagination`
- characters(limit: `Int`, page: `Int`): `CharacterPagination`

B.10.5 Subregion

Type Subregion

- id: `Int!`
- name: `String!`
- region: `Region!`
- servers(limit: `Int`, page: `Int`): `ServerPagination`

B.10.6 Zone

Type Zone

- id: `Int!`
- brackets: `Bracket`
- difficulties: `[Difficulty]`
- encounters: `[Encounter]`
- expansion: `Expansion!`
- frozen: `Boolean!`

- name: `String!`
- partitions: `[Partition]`

B.10.6.1 Bracket

- brackets
 - min: `Float!`
 - max: `Float!`
 - bucket: `Float`
 - type: `String`

B.10.6.2 Partition

- partitions
 - id: `Int!`
 - name: `String!`
 - compactName: `String!`
 - default: `Boolean!`

B.10.7 Example

B.11 Pagination Objects

B.11.1 CharacterPagination

Type CharacterPagination

- data: [Character]
- total: Int!
- per_page: Int!
- current_page: Int!
- from: Int
- to: Int
- last_page: Int!
- has_more_pages: Boolean!

B.11.2 GameAchievementPagination

Type GameAchievementPagination

- data: [GameAchievement]
- total: Int!
- per_page: Int!
- current_page: Int!
- from: Int
- to: Int
- last_page: Int!
- has_more_pages: Boolean!

B.11.3 GameAbilityPagination

Type GameAbilityPagination

- data: [GameAbility]
- total: Int!
- per_page: Int!
- current_page: Int!
- from: Int
- to: Int
- last_page: Int!
- has_more_pages: Boolean!

B.11.4 GameEnchantPagination

Type GameEnchantPagination

- data: [GameEnchant]
- total: Int!
- per_page: Int!
- current_page: Int!
- from: Int
- to: Int
- last_page: Int!
- has_more_pages: Boolean!

B.11.5 GameItemPagination

Type GameItemPagination

- data: [GameItem]
- total: Int!
- per_page: Int!
- current_page: Int!
- from: Int
- to: Int
- last_page: Int!
- has_more_pages: Boolean!

B.11.6 GameItemSetPagination

Type GameItemSetPagination

- data: [GameItemSet]
- total: Int!
- per_page: Int!
- current_page: Int!
- from: Int
- to: Int
- last_page: Int!
- has_more_pages: Boolean!

B.11.7 GameMapPagination

Type GameMapPagination

- data: [GameMap]
- total: Int!
- per_page: Int!
- current_page: Int!
- from: Int
- to: Int
- last_page: Int!
- has_more_pages: Boolean!

B.11.8 GameNPCPagination

Type GameNPCPagination

- data: [GameNPC]
- total: Int!
- per_page: Int!
- current_page: Int!
- from: Int
- to: Int
- last_page: Int!
- has_more_pages: Boolean!

B.11.9 GameZonePagination

Type GameZonePagination

- data: [GameZone]
- total: Int!
- per_page: Int!
- current_page: Int!
- from: Int
- to: Int
- last_page: Int!
- has_more_pages: Boolean!

B.11.10 GuildAttendancePagination

Type GuildAttendancePagination

- data: [GuildAttendance]
- total: Int!
- per_page: Int!
- current_page: Int!
- from: Int
- to: Int
- last_page: Int!
- has_more_pages: Boolean!

B.11.11 GuildPagination

Type GuildPagination

- data: [Guild]
- total: Int!
- per_page: Int!
- current_page: Int!
- from: Int
- to: Int
- last_page: Int!
- has_more_pages: Boolean!

B.11.12 ReportEventPaginator

Type ReportEventPaginator

- data: JSON
- nextPageTimestamp: Float

B.11.13 ReportPagination

Type ReportPagination

- data: [Report]
- total: Int!
- per_page: Int!

- current_page: Int!
- from: Int
- to: Int
- last_page: Int!
- has_more_pages: Boolean!

B.11.14 ServerPagination

Type ServerPagination

- data: [Server]
- total: Int!
- per_page: Int!
- current_page: Int!
- from: Int
- to: Int
- last_page: Int!
- has_more_pages: Boolean!

B.12 Important Types/Enums

B.12.1 CharacterRankingMetricType

enum CharacterRankingMetricType

- bossdps
- default
- dps
- hps
- krsi
- playerscore
- playerspeed
- tankhps
- wdps

B.12.2 EventDataType

enum EventDataType

- All
- Buffs
- Casts
- CombatantInfo
- DamageDone
- DamageTaken
- Deaths
- Debuffs
- Dispers
- Healing
- Interrupts
- Resources
- Summons
- Threat

B.12.3 ExternalBuffRankFilter

enum ExternalBuffRankFilter

- Any
- Require
- Exclude

B.12.4 FightRankingMetricType

enum FightRankingMetricType

- default
- execution
- feats
- score
- speed
- progress

B.12.5 GraphDataType

enum GraphDataType

- Summary
- Buffs
- Casts
- DamageDone
- DamageTaken
- Deaths
- Debuffs
- Dispel
- Healing
- Interrupts
- Resources
- Summons
- Survivability
- Threat

B.12.6 GuildRank

enum GuildRank

- NonMember
- Applicant
- Recruit
- Member
- Officer
- GuildMaster

B.12.7 HardModeLevelRankFilter

enum HardModeLevelRankFilter

- Any
- Highest
- NormalMode
- Level0
- Level1
- Level2
- Level3
- Level4

B.12.8 HostilityType

enum HostilityType

- Friendlies
- Enemies

B.12.9 KillType

```
enum KillType
    - All
    - Encounters
    - Kills
    - Trash
    - Wipes
```

B.12.10 LeaderboardRank

```
enum LeaderboardRank
    - Any
    - LogsOnly
```

B.12.11 RankingCompareType

```
enum RankingCompareType
    - Rankings
    - Parses
```

B.12.12 RankingTimeframeType

```
enum RankingTimeframeType
    - Today
    - Historical
```

B.12.13 ReportRankingMetricType

```
enum ReportRankingMetricType
    - bossdps
    - default
    - dps
    - hps
    - krsi
    - playerscore
    - playerspeed
    - tankhps
    - wdps
```

B.12.14 RoleType

```
enum RoleType
    - Any
    - DPS
    - Healer
    - Tank
```

B.12.15 SubscriptionStatus

`enum` SubscriptionStatus

- Silver
- Gold
- Platinum
- LegacySilver
- LegacyGold
- LegacyPlatinum

B.12.16 TableDataType

`enum` TableDataType

- Summary
- Buffs
- Casts
- DamageDone
- DamageTaken
- Deaths
- Debuffs
- Dispers
- Healing
- Interrupts
- Resources
- Summons
- Survivability
- Threat

B.12.17 ViewType

`enum` ViewType

- Default
- Ability
- Source
- Target

B.12.18 __DirectiveLocation

`enum` __DirectiveLocation

- QUERY
- MUTATION
- SUBSCRIPTION
- FIELD
- FRAGMENT_DEFINITION
- FRAGMENT_SPREAD
- INLINE_FRAGMENT

- VARIABLE_DEFINITION
- SCHEMA
- SCALAR
- OBJECT
- FIELD_DEFINITION
- ARGUMENT_DEFINITION
- INTERFACE
- UNION
- ENUM
- ENUM_VALUE
- INPUT_OBJECT
- INPUT_FIELD_DEFINITION

B.12.19 __TypeKind

```
enum __TypeKind
  – SCALAR
  – OBJECT
  – INTERFACE
  – UNION
  – ENUM
  – INPUT_OBJECT
  – LIST
  – NON_NULL
```

Bibliography

GraphQL. 2023. “graphql-dotnet.” <https://github.com/graphql-dotnet/graphql-dotnet>.

Warcraftlogs. 2023. “Docs.” <https://www.warcraftlogs.com/api/docs>.

WarcraftLogs and GraphQL. 2023. “GraphQL schema documentation.”
<https://www.warcraftlogs.com/v2-api-docs/warcraft/>.

Wikipedia. 2022. “Double-precision floating-point format.”
https://en.wikipedia.org/wiki/Double-precision_floating-point_format.