avrae

Andrew Zhu, D&D Beyond

Jan 27, 2021

CHEATSHEETS:

CHAPTER

ONE

GETTING STARTED

Avrae is a powerful bot, but it can be pretty daunting to get everything set up! Here's three quick steps to getting a character sheet linked with Avrae, and ready to play in a game!

1.1 Step 1: Invite Avrae to Your Server

The first step is to add Avrae to your server. Make sure you have the **Manage Server** permission, and head over to invite.avrae.io.

1.1.1 Optional: Setting a Prefix

After you add Avrae, you might want to change the prefix in case other bots use the same prefix:

```
!prefix <prefix> - Insert any prefix you want to use based on your server (ex. !, #,
↪$, !!, etc.)
```

1.1.2 Using Help

With the built in !help command, you get information about other commands in the bot. Here is the syntax for using help:

```
!help <command>
```

For example, `!help attack` will bring up the help dialog for the !attack command. Try it out for yourself!

```
!help
```

Help will give you examples of commands you can use and information about them.

1.2 Step 2: Add a Character

Once you have your stats, think of what character you want to play and make them a sheet on D&D Beyond, Dicecloud, or Google Sheets!

Once you're done making your character, make sure it's publicly viewable (Avrae needs to be able to see your sheet), grab the sharing URL, and follow the steps below depending on what sheet system you chose to use. You should see your character's stats pop up in Discord!

1.2.1 D&D Beyond

To add a character from D&D Beyond, use the following command:

```
!beyond https://ddb.ac/characters/...
```

Note: If you link your D&D Beyond and Discord accounts and your DM links your campaign to a channel, your character's rolls made on D&D Beyond or the Player App will appear in Discord!

1.2.2 Dicecloud

To add a character from Dicecloud, use the following command:

```
!dicecloud https://dicecloud.com/character/...
```

Note: Avrae can update your HP and consumables live on Dicecloud - share the sheet with edit permissions with `avrae`.

1.2.3 Google Sheets

To add a character from GSheet, use the following command:

```
!gsheet https://docs.google.com/spreadsheets/d/...
```

Note: You will need to share your sheet with `avrae-320@avrae-bot.iam.gserviceaccount.com`.

1.3 Step 3: Ready to Roll

You're ready to roll now! You can use the `!check` command to roll skill checks, `!save` for saving throws, and `!attack` to attack with your weapons!

For example:

- `!check arcana` - rolls an Intelligence (Arcana) check
- `!save dexterity` - rolls a Dexterity Save
- `!attack longsword` - rolls an attack with a longsword

1.4 Next Steps

For more detailed documentation on how each command works, you can use `!help <command>` to view a list of supported arguments, or come join us at the Avrae Development Discord!

CHAPTER

TWO

DM COMBAT GUIDE

Note: arguments surrounded like `<this>` are required, and arguments in `[brackets]` are optional. Put quotes around arguments that contain spaces.

Note: This guide will move *roughly* in chronological order, meaning commands near the top should be run first.

2.1 Starting Combat

First, combat must be started in a channel. All commands should be posted in the **same** Discord channel that combat was started in. Combat can be started by using the following command.

```
!i begin
```

Avrae will output the summary message and pin it, then a quick reminder on how to add yourself to combat (for a player).

2.1.1 Adding Monsters

After combat is started, you will need to add monsters and combatants. You can add official monsters with this command.

```
!i madd <monster name> [arguments]
```

Common arguments include:

- `-n <number of monsters>` (ex. `-n 5` adds 5 creatures)
- `-name <monster name scheme>` (ex. `-name "Orc#" -n 2` adds Orc1 and Orc2)
- `-group <group name>` (makes all creatures in the group act on the same initiative)
- `-rollhp` (rolls for a creature's HP)
- `-hp <hp>` (overrides a creature's initial HP)
- `-ac <ac>` (overrides a creature's initial AC)

Remember to surround any arguments with spaces in them with quotes!

2.1.2 Adding Other Combatants

If you're adding a combatant without a sheet, you can add a generic combatant with:

```
!i add <initiative modifier> <name> [arguments]
```

Examples of combatants that might need to be added like this are:

- an object
- a lair action
- a homebrew monster that hasn't been imported using the Bestiary system

2.1.3 Hiding Stats

As a DM, you probably want to hide certain stats from your players. By default, any monsters added with `!i madd` will have their stats hidden, but you must hide generic combatants' stats yourself:

```
!i add <initiative modifier> <name> -h
```

This will hide HP and AC.

2.1.4 Examples

```
!i madd "young red dragon"
# adds a Young Red Dragon to combat with stats hidden

!i madd kobold -n 5 -group Kobolds -rollhp
# adds 5 Kobolds, named KO1-KO5, with rolled HP, to a group named Kobolds

!i add 20 "Lair Action" -p
# adds a Lair Action on initiative 20

!i add 0 Longboat -ac 15 -hp 300
# adds an object with 300 HP, an AC of 15, and +0 initiative
```

2.2 Running Combat

Once you have finished setting up combat and your players have joined, this command will go to the next turn in the order and combat will begin.

```
!i next
```

On a player's turn, it's up to the player to run commands to take their actions. See the *Player Combat Guide*.

When a monster's turn comes up, the most common actions to take are attacking or casting a spell.

2.2.1 Attacking

To attack another combatant, use this command:

```
!i attack <attack name> -t <target name> [arguments]
```

This uses the attack list of whatever combatant's turn it is. To see a list of available attacks, run `!i attack list`.

As many targets as necessary may be provided by adding more `-t <target name>`, in the case of attacks that target multiple creatures (such as a breath weapon).

Note: If a monster makes an Attack of Opportunity, the syntax is `!i aoo <combatant name> <attack name> -t <target name> [arguments]`.

Alternatively, you may use `!ma <monster name> <attack name> -t <target name> [arguments]`.

To see all valid arguments, refer to the `!attack` and `!ma` documentation.

2.2.2 Casting a Spell

To cast a spell, use this command:

```
!i cast <spell name> [-t <target name>] [arguments]
```

This uses the spell list of whatever combatant's turn it is.

As many targets as necessary may be provided by adding more `-t <target name>`, in the case of spells that target multiple creatures (such as Fireball).

Note: If a monster casts as a reaction, the syntax is `!i rc <combatant name> <spell name> [-t <target name>] [arguments]`.

Alternatively, you may use `!mcast <monster name> <spell name> [-t <target name>] [arguments]`, although this will not track the spell slots for the monster in initiative.

To see all valid arguments, refer to the `!cast` and `!mcast` documentation.

2.2.3 Examples

```
!i attack dagger -t Caitlyn -rr 2
# attacks a player named Caitlyn with a dagger twice

!i attack longbow -t Em adv
# attacks a player named Em with a longbow at advantage

!i attack "fire breath" -t Ara -t Padellis
# makes Ara and Padellis make saves against a breath weapon

!i cast bless -t KO1 -t KO2
# casts Bless on two kobolds, and attaches an effect to automatically add 1d4

!i cast "fire bolt" -t Qal
# casts Fire Bolt at Qal
```

2.2.4 Ending A Turn

When you're done with a turn, use this command to move to the next combatant:

```
!i next
```

2.3 Helper Commands

These commands should help manually change the state of combat.

2.3.1 HP

To modify a combatant's HP:

```
!i hp <combatant name> <value>
```

To set a combatant's HP:

```
!i hp <combatant name> set <value>
```

To set a combatant's maximum HP:

```
!i hp <combatant name> max <value>
```

Examples

```
!i hp ko1 -5
# deals 5 damage to KO1

!i hp Licia set 100
# sets Licia's HP to 100

!i hp Taren max 44
# sets Taren's max HP to 44

!i hp yo1 +1d4+1
# heals YO1 for 1d4+1 HP
```

2.3.2 Attributes

To modify an attribute of a combatant:

```
!i opt <combatant name> <arguments>
```

Most common arguments:

- `-ac <AC>` (sets AC to new value)
- `-resist/immune/vuln <damage type>` (gives resistance, immunity, or vulnerability or specified type)
- `-h` (toggles whether combatants AC and HP are hidden.)

2.3.3 Effects

Effects can be used to track status effects that last a certain duration and modify a combatant's attacks, resistances, AC, or other attributes. For a full list of attributes, see `!help i effect`.

Some attacks and spells, such as Bless, will automatically add appropriate effects to their targets.

To add effects to combatants:

```
!i effect <target name> <effect name> [arguments]
```

Most common arguments:

- `-dur <duration>` (sets the duration of the effect, in rounds)
- `-b <bonus>` (adds a bonus to all of the target's attack to-hits)
- `-d <damage>` (adds bonus damage to all of the target's attacks)
- `-resist/immune/vuln <type>` (sets resistance to a damage type)

To remove Effects from combatants:

```
!i re <combatant name> [effect name]
```

Examples

```
!i effect Jozu Rage -dur 10 -d 2
# adds a Rage effect to Jozu that adds 2 damage to their attacks and lasts 10 rounds

!i effect Flore Bless -dur 10 -b 1d4 -sb 1d4
# adds a Bless effect to Flore that adds 1d4 to their attacks and saves, that lasts␣
↪10 rounds

!i effect Padellis "Mage Armor" -ac +3
# adds a Mage Armor effect to Padellis that adds 3 to their AC

!i effect Greg "Fire Shield" -resist fire -dur 1
# adds an effect to Greg that makes him resist fire until next round
```

2.4 Removing from Combat

To remove someone from combat:

```
!i remove <combatant name>
```

2.5 Ending Combat

To end combat (Avrae will ask if you wish to end combat, reply "yes"):

```
!i end
```

After combat ends, Avrae will send the person who ended it a summary of the combat.

CHAPTER

THREE

PLAYER COMBAT GUIDE

This guide will help players join combat and use actions on their turns.

Note: Arguments surrounded like `<this>` are required, and arguments in `[brackets]` are optional. Put quotes around arguments that contain spaces.

Note: This guide will move *roughly* in chronological order, meaning commands near the top should be run first.

3.1 Joining Combat

To join combat, your DM must first start it. Once they have, proceed below to the following commands:

```
!i join [arguments]
```

This will add your **active** character to combat.

Common arguments:

- `-h` (hides AC/HP)
- `adv/dis` (gives advantage/disadvantage on initiative roll)
- `-p <#>` (places at prerolled init)

You are all setup and ready to go for when your turn comes!

3.2 Your Turn

It's your turn! On your turn, the most common actions are either attacking or casting a spell:

3.2.1 Attacking

To attack, just use the same command you would use out of combat:

```
!attack <attack name> -t <target name> [arguments]
```

To see a list of your character's attacks, use `!attack list`.

As many targets as necessary may be provided by adding more `-t <target name>`, in the case of attacks that target multiple creatures (such as a breath weapon).

Note: This command will work even when it is not your turn in combat.

If you control a summoned creature, refer to the *DM Combat Guide*.

To see all valid arguments, refer to the `!attack` documentation.

3.2.2 Casting a Spell

To cast a spell, it's also the same command in and out of combat:

```
!cast <spell name> -t <target name> [arguments]
```

To see a list of your spells, use `!spellbook`.

As many targets as necessary may be provided by adding more `-t <target name>`, in the case of spells that target multiple creatures (such as Fireball).

Note: This command will work even when it is not your turn in combat.

If you control a familiar or summoned creature, refer to the *DM Combat Guide*.

To see all valid arguments, refer to the `!cast` documentation.

3.2.3 Examples

```
!attack dagger -t KO1 -rr 2
# attacks KO1 with a dagger twice

!attack longbow -t WY1 adv
# attacks WY1 with a longbow at advantage

!attack "fire breath" -t BA1 -t BA2
# makes BA1 and BA2 make saves against a breath weapon

!cast bless -t Rook -t Edmund -l 3
# casts Bless at 3rd level on Rook and Edmund, and attaches an effect to␣
↪automatically add 1d4

!cast "fire bolt" -t BA3
# casts Fire Bolt at BA3
```

3.2.4 Ending Your Turn

When you're done with your turn, use this command to move to the next combatant:

```
!i next
```

3.3 Helper Commands

These commands should help manually change the state of combat. For more reference, see the *DM Combat Guide*.

3.3.1 HP

To modify your character's HP:

```
!g hp <value>
```

To set your character's HP:

```
!g hp set <value>
```

To add temporary HP:

```
!g thp <value>
```

To set your character's maximum HP (note the different base command):

```
!i hp <character name> max <value>
```

Examples

```
!g hp -5
# deals 5 damage

!g hp set 100
# sets the character's HP to 100

!g thp 11
# gives the character 11 temp HP

!g hp +2d4+2
# heals for 2d4+2 HP
```

CHAPTER

FOUR

D&D BEYOND CONTENT INTEGRATION

With the release of Avrae 2.0, users can now Link their Discord and D&D Beyond Accounts. Here are some things that you might want to know.

4.1 How do I link my D&D Beyond and Discord accounts?

You can link your accounts on the Accounts page on D&D Beyond.

Note: It may take up to 15 minutes for Avrae to recognize the link.

To check the status, use `!ddb` to show your D&D Beyond account link.

```
!ddb
```

Linking your accounts gives you the following benefits:

4.2 Content Access

After your accounts are linked, you will be able to access any content you have purchased on D&D Beyond.

Note: If you are in a campaign with content sharing enabled you will also have access to content shared with you.

4.3 Private Character Import

Linking your D&D Beyond and Discord accounts lets Avrae import your characters from D&D Beyond without having to make the character sheet public.

4.4 Dice Sync

If your DM links your D&D Beyond campaign with a Discord channel using the `!campaign` command, any dice you roll on your character sheet or in the D&D Beyond Player App will show up in Discord in real time!

Also, any checks, saves, or attacks you roll in the linked Discord channel will appear on your character sheet and the Player App in real time!

4.5 Where can I go if I have issues or Questions?

The Avrae Development Discord is a great place to ask questions and get help where you need it. Come join us!

CHAPTER

FIVE

ALIASING BASICS

Avrae has vast potential for making long commands simple. It allows you to create and maintain commands. These commands can be used personally or shared with other users on a server. Let's take a look at some of the basics of automation that you can start using in your server.

Note: If you have experience with JSON and APIs and are looking for more advanced documentation, head on over to the Aliasing API Page.

5.1 Command Types

Avrae has a few different types of commands that are used for different purposes.

Alias - Used to shorten commands that would require a large or lengthy amount of text to use, to run code before running an Avrae command, or to write your own custom command. (In many cases, aliases are used to track features or abilities)

Examples for Alias usage:

- Short rest
- Long rest
- Sorcerer Font of Magic
- Barbarian Rage (Effects)
- Dash, Dodge, Hide Actions

Snippet - Used to augment dice rolls like saves, attacks, or ability checks.

Examples for Snippet usage:

- Guidance cantrip
- Hunter's Mark (Damage)
- Cover (3/4, Half, etc)
- Barbarian Rage (Damage)
- Bardic Inspiration

Note: In order to prevent infinite loops, aliases cannot call other aliases.

5.2 Command Levels

There are two levels of commands that are built into Avrae: user level and server level. Aliases and snippets can be setup at either level. Below is how to look at snippets or aliases at each level.

Note: If a user and a server have aliases with the same name, the user alias will take priority.

!alias - Will show user level aliases.

!servalias - Will show server level aliases.

!snippet - Will show user level snippets

!servsnippet - will show server level snippets

Note: To add server-level aliases or snippets, a user must have a role called "Dragonspeaker" or "Server Aliaser".

5.3 Help

As always you can also come to the Avrae Development Discord for help with aliasing, here.

CHAPTER

SIX

ALIASING TUTORIALS

Here are a few tutorials for aliases that were created by the Avrae Development Discord. These should take you step by step through two example aliases. Thanks to @Croebh#5603 and @silverbass#2407 for writing these, and to @Ydomat#2886 for converting them to this format!

6.1 Half-Orc Relentless Endurance Tutorial

By @silverbass#2407.

```
!alias orc-relentless
```

This sets the alias name.

```
embed
```

This is the base Avrae command, an embed, which makes the pretty text box. Check out `!help embed` for more details.

```
{{cc="Relentless Endurance"}}
```

This creates a variable for name of the custom counter, which you do need to make before using it.

```
{{v=cc_exists(cc) and get_cc(cc) and not get_hp()}}
```

This checks if the trigger conditions are valid: do you have a counter for this? is it used? are you at 0 hp?

```
-title "{{f"{name} {'uses' if v else 'tries to use'} {cc}!"}}"
```

This sets the title of the embed, to either success or fail, depending on the v variable from above. I use fstrings, or formatted strings, to streamline the code a bit.

```
-desc "{{"When you are reduced to 0 hit points but not killed outright, you can drop␣
↪to 1 hit point instead." if v else "You have more than 0 hit points." if get_hp()␣
↪else "You can't use this feature again until you finish a Long Rest." if cc_
↪exists(cc) else "You do not have this ability."}}"
```

This sets the body text of the embed, and shows the 4 cases: 1) it works, 2) you have more than 0 hp, 3) you already used the feature, 4) you don't have the counter in the first place.

```
{{mod_cc(cc, -1) if v else ""}}
```

This decrements the counter, but only if you have it. It checks this to prevent errors.

```
-f "{{f"{cc}|{cc_str(cc) if cc_exists(cc) else '*None*'}"}}"
```

This displays the counter, or None if you don't have it. It's displayed in the embed as a field. Again, using an fstring for streamlined code.

```
{{set_hp(1) if v and not get_hp() else ""}}
```

This sets your hit points to 1, but only if you have 0 right now.

```
-color <color> -thumb <image>
```

This makes it look pretty.

The end result is:

```
!alias orc-relentless embed {{cc="Relentless Endurance"}} {{v=cc_exists(cc) and get_
↪cc(cc) and not get_hp()}} -title "{{f"{name} {'uses' if v else 'tries to use'} {cc}!
↪"}}" -desc "{{"When you are reduced to 0 hit points but not killed outright, you␣
↪can drop to 1 hit point instead." if v else "You have more than 0 hit points." if␣
↪get_hp() else "You can't use this feature again until you finish a Long Rest." if␣
↪cc_exists(cc) else "You do not have this ability."}}" {{mod_cc(cc, -1) if v else ""}
↪} -f "{{f"{cc}|{cc_str(cc) if cc_exists(cc) else '*None*'}"}}" {{set_hp(1) if v and␣
↪not get_hp() else ""}} -color <color> -thumb <image>
```

6.2 Insult Tutorial

By @Croebh#5603

```
!servalias insult embed
```

This creates a servalias named insult, calling the command embed.

```
{{ G = get_gvar("68c31679-634d-46de-999b-4e20b1f8b172") }}
```

This sets a local variable, G to the contents of the gvar with the ID 68c31679-634d-46de-999b-4e20b1f8b172. The get_gvar() function grabs the content of the Gvar as plain text.

```
{{ L = [x.split(",") for x in G.split("\n\n")] }}
```

This sets a local variable, L to a list comprehension. What that is doing is breaking down the variable G into a list of lists.

`G.split("\n\n")`

So, this is splitting text everytime there is two line breaks. In this case, it ends up being in three parts.

`x.split(",") for x in`

This part is saying for each part of the split we did above, call that part x, then split THAT part on every comma. So L ends up being something like `[["Words","Stuff"],["Other","Words","More!"],["More","Words"]]`

```
{{ I = [x.pop(roll(f'1d{len(x)}-1')).title() for x in L] }}
```

This sets another local variable, I, to another list comprehension, this time iterating on the variable L.

`x.pop(roll(f'1d{len(x)}-1')).title()`

Okay, a little more complicated. We're going to start in the middle.

`f'1d{len(x)}-1'`

So, this is an f-string, or formatted strings. It allows us to run code in the middle of string, in this case `{len(x)}`, which will be the length of x (which is the current part of L that we're looking at.). So in our example, say we're looking at the first part of L, which is `["words","stuff"]`. The length of this is 2, so it will return the string, `1d2-1`. The -1 is important because lists are 0-indexed, that is, the first item in the list has an index of 0 (as opposed to 1).

`roll()`

This rolls the returned string, which as we determined above, is 1d2-1. Lets say it returns 1.

`x.pop()`

What this does is pop the item at the given index out of the list. This removes the item from the list, and returns it. This removes the chance of that particular item being chosen again. With our result of 1, this will return the second item (because its index-0), which is `stuff`. This will make x be `["words"]` now.

`.title()`

This just capitalizes the first character of each word in the string. Now it will return `Words`

Now, iterating over this list could make I `["Words","More!","Words"]`, and those would be removed from L, so L is now `[["stuff"],["Other","Words"],["More"]]`

```
{{ aL = L[0] + L[1] }}
```

This sets the variable aL to the combination of the first results of L, so `["stuff"]` and `["Other","Words"]`, making aL `["stuff","Other","Words"]`, as they were added together. This doesn't remove those two lists from L

```
{{ add = [aL.pop(roll(f'1d{len(aL)-1}')).title() for x in range(int("&1&".strip("&
↪")))]}}
```

Another fun one. This sets the variable `add` to another list comprehension, this time on a varible list.

`range(int("&1&".strip("&")))`

`&1&` is a placeholder that gets replaced by the first argument given to the alias. So with `!insult 3`, `&1&` would return `3`. However, with no args given, it doesn't get replaced, and stays as `&1&`.

`.strip('&')`

So, this strips the '&' character from either side of the string. This lets us have a default of "1" when no arguments given (because "&1&" with the "&"'s removed is "1")

`int()`

this converts the string to a integer. This will error if the first arg is anything other than a number (like if anyone were to `!insult silverbass`)

`range()`

This creates a list of numbers. In this case, because only one argument is given to it, it creates a list of numbers from 0 to the number given, not including that number. So with an argument of 1, it will make a list `[0]`, but with an argument of 3, it will return `[0,1,2]`

`aL.pop(roll(f'1d{len(aL)-1}')).title()`

More fun, but its basically the exact same as the last time. A formatted string, this time calling the length of the aL list as opposed to the current iteration. A roll of that string, and then a pop out of aL, returning and removing the given index, then capitalizing it.

For this example, lets say the user did `!insult 2`. So the range will return `[0,1]`, making it do the function twice. The length of aL the first time is 3, so it will roll 1d3-1, let's say it returns 0. This will get popped out of aL as "Stuff"

The second time it runs, the length is 2 (because we just removed one result), so it will roll 1d2-1. This time lets say we got 1, so the second time it will return "Words".

So add is now `["Stuff", "Words"]`

```
{{ I = [I[0], I[1]] + add + [I[2]] }}
```

This overwrites the variable I with a new list.

`[I[0], I[1]]`

So this will be the first two items in I, `"Words" and "More!"`, making it `["Words","More!"]`.

`add` is just the entire add variable, `["Stuff", "Words"]`

And finally, `[I[2]]` is the third (and final) item in I, `"Words"`

Combining them all together, the variable I is now, `["Words","More!","Stuff", "Words","Words"]`

```
-title "You {{" ".join(I)}}!"
```

So, this adds a -title to the embed the command starts with. The contents of this title is `"You {{" ".join(I)}}!"`

`{{" ".join(I)}}`

This joins the contents of the variable I, putting space (" ") between each item. So in this case, it would return `"Words More! Stuff Words Words"`

Putting that together with the text outside the code, the title will be `"You Words More! Stuff Words Words!"`

```
-thumb <image> -color <color>
```

This just sets the thumbnail and color of the embed to those that are set on your character.

The end result is:

```
!servalias insult embed
{{ G = get_gvar("68c31679-634d-46de-999b-4e20b1f8b172") }}
{{ L = [x.split(",") for x in G.split("\n\n")] }}
{{ I = [x.pop(roll(f'1d{len(x)}-1')).title() for x in L] }}
{{ aL = L[0] + L[1] }}
{{ add = [aL.pop(roll(f'1d{len(aL)-1}')).title() for x in range(int("&1&".strip("&
↪")))]}}
{{ I = [I[0], I[1]] + add + [I[2]] }}
-title "You {{" ".join(I)}}!"
-thumb <image> -color <color>
```

CHAPTER

SEVEN

ALIASING API

So you want to write aliases for your commonly used commands - cool! This cheatsheet details some of the nitty-gritty syntactical shenanigans that you can use to make your aliases very powerful.

When placed inline in an alias, any syntax in the syntax table will have the listed effect. For a list of built-in cvars, see the *Cvar Table*.

For a list of user-created aliases, plus help aliasing, join the Avrae Discord!

7.1 Draconic

The language used in Avrae aliases is a custom modified version of Python, called Draconic. In most cases, Draconic uses the same syntax and base types as Python - any exceptions will be documented here!

Note: It is highly recommended to be familiar with the Python language before diving into Draconic, as the two use the same syntax and types.

As Draconic is meant to both be an operational and templating language, there are multiple ways to use Draconic inside your alias.

7.2 Syntax

This section details the special syntax used in the Draconic language. Note that these syntaxes are only evaluated in an alias, the `test` command, or the `tembed` command.

7.2.1 Rolls

Syntax: `{diceexpr}`

Description: Rolls the expression inside the curly braces and is replaced by the result. If an error occurs, is replaced by `0`. Variables are allowed inside the expression.

Examples

```
>>> !test Rolling 1d20: {1d20}
Rolling 1d20: 7
```

```
>>> !test Strength check: {1d20 + strengthMod}
Strength check: 21
```

7.2.2 Values

Syntax: `<var>`

Description: Replaced by the value of the variable, implicitly cast to `str`. The variable can be a user variable, character variable, or a local variable set in a Draconic script.

Examples

```
>>> !test My strength modifier is: <strengthMod>
My strength modifier is: 2
```

7.2.3 Draconic Expressions

Syntax: `{{code}}`

Description: Runs the Draconic code inside the braces and is replaced by the value the code evaluates to. If the code evaluates to `None`, is removed from the output.

See below for a list of builtin Draconic functions.

Examples

```
>>> !test 1 more than my strength score is {{strength + 1}}!
1 more than my strength score is 15!
```

```
>>> !test My roll was {{"greater than" if roll("1d20") > 10 else "less than"}} 10!
My roll was less than 10!
```

7.2.4 Draconic Blocks

Syntax

```
<drac2>
code
</drac2>
```

Description: Runs the multi-line Draconic code between the delimiters. If a value is returned (via the `return` keyword), is replaced by the returned value.

Examples

```
>>> !test <drac2>
... out = []
... for i in range(5):
...   out.append(i * 2)
...   if i == 2:
...     break
... return out
... </drac2>
[0, 2, 4]
```

```
>>> !test <drac2>
... out = []
... for stat in ['strength', 'dexterity', 'constitution']:
```

(continues on next page)

(continued from previous page)

```
...   out.append(get(stat))
... </drac2>
... My STR, DEX, and CON scores are {{out}}!
My STR, DEX, and CON scores are [12, 18, 14]!
```

7.2.5 Argument Parsing

Often times when writing aliases, you will need to access user input. These special strings will be replaced with user arguments (if applicable)!

Syntax: `%1%`, `%2%`, etc.

Description: Replaced with the Nth argument passed to the alias. If the argument contains spaces, the replacement will contain quotes around the argument.

Syntax: `%*%`

Description: Replaced with the unmodified string following the alias.

Syntax: `&1&`, `&2&`, etc.

Description: Replaced with the Nth argument passed to the alias. If the argument contains spaces, the replacement will **not** contain quotes around the argument. Additionally, any quotes in the argument will be backslash-escaped.

Syntax: `&*&`

Description: Replaced with the string following the alias. Any quotes will be backslash-escaped.

Syntax: `&ARGS&`

Description: Replaced with a list representation of all arguments - usually you'll want to put this in Draconic code.

Examples

```
>>> !alias asdf echo %2% %1%
>>> !asdf first "second arg"
"second arg" first
```

```
>>> !alias asdf echo %*% first
>>> !asdf second "third word"
second "third word" first
```

```
>>> !alias asdf echo &1& was the first arg
>>> !asdf "hello world"
hello world was the first arg
```

```
>>> !alias asdf echo &*& words
>>> !asdf second "third word"
second \"third word\" words
```

```
>>> !alias asdf echo &ARGS&
>>> !asdf first "second arg"
['first', 'second arg']
```

7.3 Cvar Table

This table lists the available cvars when a character is active.

Name	Description	Type
armor	Armor Class.	int
charisma	Charisma score.	int
charismaMod	Charisma modifier.	int
charismaSave	Charisma saving throw modifier.	int
constitution	Constitution score.	int
constitutionMod	Constitution modifier.	int
constitutionSave	Constitution saving throw modifier.	int
description	Full character description.	str
dexterity	Dexterity score.	int
dexterityMod	Dexterity modifier.	int
dexteritySave	Dexterity saving throw modifier.	int
hp	Maximum hit points.	int
image	Character image URL.	str
intelligence	Intelligence score.	int
intelligenceMod	Intelligence modifier.	int
intelligenceSave	Intelligence saving throw modifier.	int
level	Character level.	int
name	The character's name.	str
proficiencyBonus	Proficiency bonus.	int
spell	The character's spellcasting ability mod.	int
strength	Strength score.	int
strengthMod	Strength modifier.	int
strengthSave	Strength saving throw modifier.	int
wisdom	Wisdom score.	int
wisdomMod	Wisdom modifier.	int
wisdomSave	Wisdom saving throw modifier.	int
XLevel	How many levels a character has in class X.	int

Note: `XLevel` is not guaranteed to exist for any given `X`, and may not exist for GSheet 1.3/1.4 characters. It is recommended to use `AliasCharacter.levels.get()` to access arbitrary levels instead.

7.4 Function Reference

Warning: It may be possible to corrupt your character data by incorrectly calling functions. Script at your own risk.

7.4.1 All Contexts

These functions are available in any scripting context, regardless if you have a character active or not.

Python Builtins

all (*iterable*)
Return `True` if all elements of the *iterable* are true, or if the iterable is empty.

any (*iterable*)
Return `True` if any element of the *iterable* is true. If the iterable is empty, return `False`.

ceil (*x*)
Rounds a number up to the nearest integer. See `math.ceil()`.

Parameters **x** (*float or int*) – The number to round.

Returns The smallest integer >= x.

Return type int

enumerate (*x*)
Returns a iterable of tuples containing a count and the values from the iterable.

Parameters **x** (*iterable*) – The value to convert.

Returns enumerate of count and objects.

Return type iterable[tuple[int, any]]

float (*x*)
Converts *x* to a floating point number.

Parameters **x** (*str, int, or float*) – The value to convert.

Returns The float.

Return type float

floor (*x*)
Rounds a number down to the nearest integer. See `math.floor()`.

Parameters **x** (*float or int*) – The number to round.

Returns The largest integer <= x.

Return type int

int (*x*)
Converts *x* to an integer.

Parameters **x** (*str, int, or float*) – The value to convert.

Returns The integer.

Return type int

len (*s*)
Return the length (the number of items) of an object. The argument may be a sequence (such as a string, bytes, tuple, list, or range) or a collection (such as a dictionary, set, or frozen set).

Returns The length of the argument.

Return type int

max (*iterable*, *[, *key*, *default*])
max (*arg1*, *arg2*, **args*[, *key*])
Return the largest item in an iterable or the largest of two or more arguments.

If one positional argument is provided, it should be an iterable. The largest item in the iterable is returned. If two or more positional arguments are provided, the largest of the positional arguments is returned.

There are two optional keyword-only arguments. The key argument specifies a one-argument ordering function like that used for `list.sort()`. The default argument specifies an object to return if the provided iterable is empty. If the iterable is empty and default is not provided, a `ValueError` is raised.

If multiple items are maximal, the function returns the first one encountered.

min (*iterable*, ***[, *key*, *default*])
min (*arg1*, *arg2*, **args*[, *key*])

Return the smallest item in an iterable or the smallest of two or more arguments.

If one positional argument is provided, it should be an iterable. The smallest item in the iterable is returned. If two or more positional arguments are provided, the smallest of the positional arguments is returned.

There are two optional keyword-only arguments. The key argument specifies a one-argument ordering function like that used for `list.sort()`. The default argument specifies an object to return if the provided iterable is empty. If the iterable is empty and default is not provided, a `ValueError` is raised.

If multiple items are minimal, the function returns the first one encountered.

range (*stop*)
range (*start*, *stop*[, *step*])

Returns a list of numbers in the specified range.

If the step argument is omitted, it defaults to `1`. If the start argument is omitted, it defaults to `0`. If step is zero, `ValueError` is raised.

For a positive step, the contents of a range `r` are determined by the formula `r[i] = start + step*i` where `i >= 0` and `r[i] < stop`.

For a negative step, the contents of the range are still determined by the formula `r[i] = start + step*i`, but the constraints are `i >= 0` and `r[i] > stop`.

A range object will be empty if r[0] does not meet the value constraint. Ranges do support negative indices, but these are interpreted as indexing from the end of the sequence determined by the positive indices.

Parameters

- **start** (*int*) – The start of the range (inclusive).
- **stop** (*int*) – The end of the range (exclusive).
- **step** (*int*) – The step value.

Returns The range of numbers.

Return type list

round (*number*[, *ndigits*])

Return number rounded to ndigits precision after the decimal point. If ndigits is omitted or is None, it returns the nearest integer to its input.

Parameters

- **number** (*float or int*) – The number to round.
- **ndigits** (*int*) – The number of digits after the decimal point to keep.

Returns The rounded number.

Return type float

sqrt (*x*)

See `math.sqrt()`.

Returns The square root of *x*.

Return type float

str(*x*)
Converts *x* to a string.

Parameters **x** (*Any*) – The value to convert.

Returns The string.

Return type str

sum(*iterable*[, *start*])
Sums *start* and the items of an *iterable* from left to right and returns the total. *start* defaults to `0`. The *iterable*'s items are normally numbers, and the start value is not allowed to be a string.

time()
Return the time in seconds since the UNIX epoch (Jan 1, 1970, midnight UTC) as a floating point number. See `time.time()`.

Returns The epoch time.

Return type float

Draconic Functions

argparse(*args*)
Parses arguments.

Parameters **args** (*str or Iterable*) – A list of arguments to parse.

Returns The parsed arguments.

Return type *ParsedArguments*

```
>>> args = argparse("adv -rr 2 -b 1d4[bless]")
>>> args.adv()
1
>>> args.last('rr')
'2'
>>> args.get('b')
['1d4[bless]']
```

character()
Returns the active character if one is. Otherwise, raises a `FunctionRequiresCharacter` error.

Return type *AliasCharacter*

combat()
Returns the combat active in the channel if one is. Otherwise, returns `None`.

Return type *SimpleCombat*

Note: If called outside of a character context, `combat().me` will be `None`.

---

ctx
The context the alias was invoked in. See *AliasContext* for more details.

Note that this is an automatically bound name and not a function.

Type *AliasContext*

delete_uvar(*name*)
Deletes a user variable. Does nothing if the variable does not exist.

Parameters **name** (*str*) – The name of the variable to delete.

dump_json(*self*, *obj*)
Serializes an object to a JSON string. See `json.dumps()`.

err(*reason*, *pm_user=False*)
Stops evaluation of an alias and shows the user an error.

Parameters

- **reason** (*str*) – The error to show.
- **pm_user** (*bool*) – Whether or not to PM the user the error traceback.

Raises AliasException

exists(*name*)
Returns whether or not a name is set in the current evaluation context.

Return type bool

get(*name*, *default=None*)
Gets the value of a name, or returns *default* if the name is not set.

Retrieves names in the order of local > cvar > uvar. Does not interact with svars.

Parameters

- **name** (*str*) – The name to retrieve.
- **default** – What to return if the name is not set.

get_gvar(*address*)
Retrieves and returns the value of a gvar (global variable).

Parameters **address** (*str*) – The gvar address.

Returns The value of the gvar.

Return type str

get_svar(*name*)
Retrieves and returns the value of a svar (server variable).

Parameters

- **name** (*str*) – The name of the svar.
- **default** – What to return if the name is not set.

Returns The value of the svar, or the default value if it does not exist.

Return type str or None

load_json(*self*, *jsonstr*)
Loads an object from a JSON string. See `json.loads()`.

randint(*x*)
Returns a random integer in the range `[0..x)`.

Parameters **x** (*int*) – The upper limit (non-inclusive).

Returns A random integer.

Return type int

roll(*dice*)
Rolls dice and returns the total.

Parameters **dice** (*str*) – The dice to roll.

Returns The roll's total, or 0 if an error was encountered.

Return type int

set_uvar(*name*, *value*)
Sets a user variable.

Parameters

- **name** (*str*) – The name of the variable to set.
- **value** (*str*) – The value to set it to.

set_uvar_nx(*name*, *value*)
Sets a user variable if there is not already an existing name.

Parameters

- **name** (*str*) – The name of the variable to set.
- **value** (*str*) – The value to set it to.

typeof(*inst*)
Returns the name of the type of an object.

Parameters **inst** – The object to find the type of.

Returns The type of the object.

Return type str

uvar_exists(*name*)
Returns whether a uvar exists.

Return type bool

vroll(*rollStr*, *multiply=1*, *add=0*)
Rolls dice and returns a detailed roll result.

Parameters

- **dice** (*str*) – The dice to roll.
- **multiply** (*int*) – How many times to multiply each set of dice by.
- **add** (*int*) – How many dice to add to each set of dice.

Returns The result of the roll.

Return type *SimpleRollResult*

Warning: The following functions are deprecated and should be avoided:

chanid()
Returns the ID of the active Discord channel.

Deprecated since version 2.5.0: Use `ctx.channel.id` instead.

Return type str

servid()
Returns the ID of the active Discord guild, or None if in DMs.

Deprecated since version 2.5.0: Use `ctx.guild.id` instead.

Return type str

set(*name*, *value*)
Sets the value of a name in the current scripting context.

Deprecated since version 0.1.0: Use `name = value` instead.

Parameters

- **name** – The name to set.
- **value** – The value to set it to.

7.4.2 Character Context

These functions are only available when a character is active in a scripting context. Otherwise, attempts to call these functions will raise a `FunctionRequiresCharacter` exception.

Warning: As of v2.2.0, *all* character context functions have been deprecated and should be replaced. See each function's documentation for its replacement.

Custom Counters

Warning:

cc_exists(*name*)
Deprecated since version 2.5.0: Use `character().cc_exists()` instead.

Returns whether a custom counter exists.

Parameters **name** (*str*) – The name of the custom counter to check.

Returns Whether the counter exists.

Return type bool

cc_str(*name*)
Deprecated since version 2.5.0: Use `character().cc_str()` instead.

Returns a string representing a custom counter.

Parameters **name** (*str*) – The name of the custom counter to get.

Returns A string representing the current value, maximum, and minimum of the counter.

Return type str

Raises `ConsumableException` if the counter does not exist.

Example:

```
>>> cc_str("Ki")
'11/17'
>>> cc_str("Bardic Inspiration")
''
```

create_cc(*name*, *minVal=None*, *maxVal=None*, *reset=None*, *dispType=None*)
Deprecated since version 2.5.0: Use `character().create_cc()` instead.

Creates a custom counter. If a counter with the same name already exists, it will replace it.

Parameters

- **name** (*str*) – The name of the counter to create.
- **minVal** (*str*) – The minimum value of the counter. Supports *Cvar Table* parsing.
- **maxVal** (*str*) – The maximum value of the counter. Supports *Cvar Table* parsing.
- **reset** (*str*) – One of `'short'`, `'long'`, `'hp'`, `'none'`, or `None`.
- **dispType** (*str*) – Either `None` or `'bubble'`.

create_cc_nx(*name*, *minVal=None*, *maxVal=None*, *reset=None*, *dispType=None*)
Deprecated since version 2.5.0: Use `character().create_cc_nx()` instead.

Creates a custom counter if one with the given name does not already exist. Equivalent to:

```
>>> if not cc_exists(name):
>>>     create_cc(name, minVal, maxVal, reset, dispType)
```

delete_cc(*name*)
Deprecated since version 2.5.0: Use `character().delete_cc()` instead.

Deletes a custom counter.

Parameters **name** (*str*) – The name of the custom counter to delete.

Raises `ConsumableException` if the counter does not exist.

get_cc(*name*)
Deprecated since version 2.5.0: Use `character().get_cc()` instead.

Gets the value of a custom counter.

Parameters **name** (*str*) – The name of the custom counter to get.

Returns The current value of the counter.

Return type int

Raises `ConsumableException` if the counter does not exist.

get_cc_max(*name*)
Deprecated since version 2.5.0: Use `character().get_cc_max()` instead.

Gets the maximum value of a custom counter.

Parameters **name** (*str*) – The name of the custom counter maximum to get.

Returns The maximum value of the counter. If a counter has no maximum, it will return an obscenely large number (2^31-1).

Return type int

Raises `ConsumableException` if the counter does not exist.

get_cc_min(*name*)
Deprecated since version 2.5.0: Use `character().get_cc_min()` instead.

Gets the minimum value of a custom counter.

Parameters **name** (*str*) – The name of the custom counter minimum to get.

Returns The minimum value of the counter. If a counter has no minimum, it will return an obscenely small number (-2^31).

Return type int

Raises `ConsumableException` if the counter does not exist.

mod_cc(*name*, *value*, *strict=False*)
Deprecated since version 2.5.0: Use `character().mod_cc()` instead.

Modifies the value of a custom counter. Equivalent to `set_cc(name, get_cc(name) + value, strict)`.

set_cc(*name*, *value*, *strict=False*)
Deprecated since version 2.5.0: Use `character().set_cc()` instead.

Sets the value of a custom counter.

Parameters

- **name** (*str*) – The name of the custom counter to set.
- **value** (*int*) – The value to set the counter to.
- **strict** (*bool*) – If `True`, will raise a `CounterOutOfBounds` if the new value is out of bounds, otherwise silently clips to bounds.

Raises `ConsumableException` if the counter does not exist.

Spell Slots

Warning:

get_slots(*level*)
Deprecated since version 2.5.0: Use `character().spellbook.get_slots()` instead.

Gets the number of remaining spell slots of a given level that a character has.

Parameters **level** (*int*) – The level to get the remaining slots of.

Returns The number of remaining slots of that level.

Return type int

get_slots_max(*level*)
Deprecated since version 2.5.0: Use `character().spellbook.get_slots_max()` instead.

Gets the maximum number of spell slots of a given level that a character has.

Parameters **level** (*int*) – The level to get the maximum slots of.

Returns The maximum number of slots of that level.

Return type int

set_slots(*level*, *value*)
Deprecated since version 2.5.0: Use `character().spellbook.set_slots()` instead.

Sets how many spell slots of a given level a character has.

Parameters

- **level** (*int*) – The level of spell slots to set.

- **value** (*int*) – The value to set the remaining slots to.

Raises `CounterOutOfBounds` if the number of slots is invalid.

slots_str(*level*)
Deprecated since version 2.5.0: Use `character().spellbook.slots_str()` instead.

Returns a string representing how many spell slots a character has of a given level.

Parameters **level** (*int*) – The level to get the slots of.

Returns A string representing the current remaining and maximum number of slots of that level.

Return type str

use_slot(*level*)
Deprecated since version 2.5.0: Use `character().spellbook.use_slot()` instead.

Uses one spell slot of a given level. Equivalent to `set_slots(level, get_slots(level) - 1)`.

Hit Points

Warning:

get_hp()
Deprecated since version 2.5.0: Use `character().hp` instead.

Returns The character's current hit points.

Return type int

get_temphp()
Deprecated since version 2.5.0: Use `character().temp_hp` instead.

Returns The character's current temporary hit points.

Return type int

hp_str()
Deprecated since version 2.5.0: Use `character().hp_str()` instead.

Returns a string representing a character's current HP, max HP, and temp HP.

mod_hp(*value*, *overflow=True*)
Deprecated since version 2.5.0: Use `character().modify_hp()` instead.

Modifies the character's remaining hit points by *value*. If *value* is negative, will deal damage to temp HP first.

Parameters

- **value** (*int*) – How much to modify remaining HP by.
- **overflow** (*bool*) – If `False`, clips the new HP value to `[0..hp]`.

set_hp(*value*)
Deprecated since version 2.5.0: Use `character().set_hp()` instead.

Sets the character's remaining hit points. Ignores temp HP.

Parameters **value** (*int*) – The new value for hit points.

set_temphp(*value*)
Deprecated since version 2.5.0: Use `character().set_temp_hp()` instead.

Sets the character's remaining temp HP.

Parameters **value** (*int*) – The new value for temporary hit points.

Cvars

Warning:

Note: All custom character variables are locally bound to a Draconic scope. To access their values, use them like a normal name.

---

delete_cvar(*name*)
Deprecated since version 2.5.0: Use `character().delete_cvar()` instead.

Deletes a custom character variable. Does nothing if the cvar does not exist.

Parameters **name** (*str*) – The name of the variable to delete.

set_cvar(*name*, *value*)
Deprecated since version 2.5.0: Use `character().set_cvar()` instead.

Sets a custom character variable, which will be available in all scripting contexts using this character.

Parameters

- **name** (*str*) – The name of the variable to set. Must be a valid identifier and not be in the *Cvar Table*.
- **value** (*str*) – The value to set it to.

set_cvar_nx(*name*, *value*)
Deprecated since version 2.5.0: Use `character().set_cvar_nx()` instead.

Sets a custom character variable if it is not already set.

Parameters

- **name** (*str*) – The name of the variable to set. Must be a valid identifier and not be in the *Cvar Table*.
- **value** (*str*) – The value to set it to.

Other

Warning:

get_raw()
Deprecated since version 2.5.0: Deprecated without replacement. Use `character()` to get a representation of the character instead.

Returns a raw representation of character data.

7.5 Variable Scopes

In addition to Python's normal variable scoping rules, Avrae introduces 4 new scopes in the form of character variables, user variables, server variables, and global variables. The intended purpose and binding rules of each are detailed below.

Variable Type	Read	Write	Binding	Scope	Who
Cvar	Yes	Yes	Implicit	Character	User
Uvar	Yes	Yes	Implicit	User	User
Svar	Yes	No	Explicit	Server	Anyone on server
Gvar	Yes	No	Explicit	Everywhere	Anyone

7.5.1 Character Variables

aka cvars

Character variables are variables bound to a character. These are usually used to set character-specific defaults or options for aliases and snippets (e.g. a character's familiar type/name). When running an alias or snippet, cvars are *implicitly* bound as local variables in the runtime.

Cvars can be written or deleted in Draconic using `AliasCharacter.set_cvar()` and `AliasCharacter.delete_cvar()`, respectively.

All characters contain some built-in character variables (see *Cvar Table*). These cannot be overwritten.

7.5.2 User Variables

aka uvars

User variables are bound per Discord user, and will go with you regardless of what server or character you are on. These variables are usually used for user-specific options (e.g. a user's timezone, favorite color, etc.). When running an alias or snippet, uvars are *implicitly* bound as local variables in the runtime. If a cvar and uvar have the same name, the cvar takes priority.

Uvars can be written or deleted in Draconic using `set_uvar()` or `delete_uvar()`, respectively.

7.5.3 Server Variables

aka svars

Server variables are named variables bound per Discord server, and can only be accessed in the Discord server they are bound in. These variables are usually used for server-specific options for server aliases (e.g. stat rolling methods, server calendar, etc.). Unlike cvars and uvars, svars must be *explicitly* retrieved in an alias by calling `get_svar()`. Svars can be listed and read by anyone on the server, so be careful what data you store!

Svars can only be written or deleted using `!svar <name> <value>` and `!svar delete <name>`, respectively. These commands can only be issued by a member who has Administrator Discord permissions or a Discord role named "Server Aliaser" or "Dragonspeaker".

7.5.4 Global Variables

aka gvars

Global variables are uniquely named variables that are accessible by anyone, anywhere in Avrae. These variables are usually used for storing large amounts of read-only data (e.g. an alias' help message, a JSON file containing cards, etc.). These variables are automatically assigned a unique name on creation (in the form of a 36 character UUID), and must be *explicitly* retrieved in an alias by calling *get_gvar()*. Gvars can be read by anyone, so be careful what data you store!

Gvars can only be created using `!gvar create <value>`, and by default can only be edited by its creator. See `!help gvar` for more information.

7.6 See Also

Draconic's syntax is very similar to Python. Other Python features supported in Draconic include:

- Ternary Operators (`x if a else y`)
- Slicing (`"Hello world!"[2:4]`)
- Operators (`2 + 2`, `"foo" in "foobar"`, etc)
- Assignments (`a = 15`)
- List Comprehensions

7.7 Initiative Models

7.7.1 SimpleCombat

class SimpleCombat

combatants
A list of all *SimpleCombatant* in combat.

current
The *SimpleCombatant* or *SimpleGroup* representing the combatant whose turn it is.

groups
A list of all *SimpleGroup* in combat.

me
The *SimpleCombatant* representing the active character in combat, or `None` if the character is not in the combat.

round_num
An *int* representing the round number of the combat.

turn_num
An *int* representing the initiative score of the current turn.

get_combatant(*name*)
Gets a *SimpleCombatant*, fuzzy searching (partial match) on name.

Parameters **name** (*str*) – The name of the combatant to get.

Returns The combatant.

Return type *SimpleCombatant*

get_group(*name*)
Gets a *SimpleGroup*, fuzzy searching (partial match) on name.

Parameters **name** (*str*) – The name of the group to get.

Returns The group.

Return type *SimpleGroup*

7.7.2 SimpleCombatant

class SimpleCombatant(*AliasStatBlock*)
Represents a combatant in combat.

effects
A list of *SimpleEffect* active on the combatant.

Type list of *SimpleEffect*

init
What the combatant rolled for initiative.

Type int

initmod
An int representing the combatant's initiative modifier.

Type int

level
Deprecated since version 2.5.0: Use `SimpleCombatant.levels.total_level` or `SimpleCombatant.spellbook.caster_level` instead.

The combatant's spellcaster level. `0` if the combatant is not a player or spellcaster.

Type int

resists
Deprecated since version 2.5.0: Use `SimpleCombatant.resistances` instead.

The combatant's resistances, immunities, and vulnerabilities.

Type *AliasResistances*

type
The type of the object (`"combatant"`), to determine whether this is a group or not.

Type str

property ac
The armor class of the creature.

Return type int or None

add_effect(*name: str*, *args: str*, *duration: int = -1*, *concentration: bool = False*, *parent=None*, *end: bool = False*, *desc: str = None*)
Adds an effect to the combatant.

Parameters

- **name** (*str*) – The name of the effect to add.
- **args** (*str*) – The effect arguments to add (same syntax as init effect).
- **duration** (*int*) – The duration of the effect, in rounds.

- **concentration** (*bool*) – Whether the effect requires concentration.
- **parent** (*SimpleEffect*) – The parent of the effect.
- **end** (*bool*) – Whether the effect ticks on the end of turn.
- **desc** (*str*) – A description of the effect.

property attacks
The attacks of the creature.

Return type *AliasAttackList*

property controller
The ID of the combatant's controller.

Return type int

damage(*dice_str*, *crit=False*, *d=None*, *c=None*, *critdice=0*, *overheal=False*)
Does damage to a combatant, and returns the rolled result and total, accounting for resistances.

Parameters

- **dice_str** (*str*) – The damage to do (e.g. `"1d6[acid]"`).
- **crit** (*bool*) – Whether or not the damage should be rolled as a crit.
- **d** (*str*) – Any additional damage to add (equivalent of -d).
- **c** (*str*) – Any additional damage to add to crits (equivalent of -c).
- **critdice** (*int*) – How many extra weapon dice to roll on a crit (in addition to normal dice).
- **overheal** (*bool*) – Whether or not to allow this damage to exceed a target's HP max.

Returns Dictionary representing the results of the Damage Automation.

Return type dict

get_effect(*name: str*)
Gets a SimpleEffect, fuzzy searching (partial match) for a match.

Parameters **name** (*str*) – The name of the effect to get.

Returns The effect.

Return type *SimpleEffect*

property group
The name of the group the combatant is in, or `None` if the combatant is not in a group.

Return type str or None

property hp
The current HP of the creature.

Return type int or None

hp_str()
Returns a string describing the creature's current, max, and temp HP.

Return type str

property levels
The levels of the creature.

Return type *AliasLevels*

property max_hp
The maximum HP of the creature.

Return type int or None

property maxhp
Deprecated since version 2.5.0: Use `SimpleCombatant.max_hp` instead.

The combatant's maximum hit points. `None` if not set.

Return type Optional[int]

mod_hp(*mod: int*, *overheal: bool = False*)
Deprecated since version 2.5.0: Use `SimpleCombatant.modify_hp()` instead.

Modifies a combatant's remaining hit points by a value.

Parameters

- **mod** (*int*) – The amount of HP to add.
- **overheal** (*bool*) – Whether to allow exceeding max HP.

modify_hp(*amount*, *ignore_temp=False*, *overflow=True*)
Modifies the creature's remaining HP by a given amount.

Parameters

- **amount** (*int*) – The amount of HP to add/remove.
- **ignore_temp** (*bool*) – If *amount* is negative, whether to damage temp HP first or ignore temp.
- **overflow** (*bool*) – If *amount* is positive, whether to allow overhealing or cap at the creature's max HP.

property name
The name of the creature.

Return type str

property note
The note on the combatant. `None` if not set.

Return type str or None

remove_effect(*name: str*)
Removes an effect from the combatant, fuzzy searching on name. If not found, does nothing.

Parameters **name** (*str*) – The name of the effect to remove.

reset_hp()
Heals a creature to max and removes any temp HP.

property resistances
The resistances, immunities, and vulnerabilities of the creature.

Return type *AliasResistances*

save(*ability: str*, *adv: bool = None*)
Rolls a combatant's saving throw.

Parameters

- **ability** (*str*) – The type of save ("str", "dexterity", etc).

- **adv** (*bool*) – Whether to roll the save with advantage. Rolls with advantage if `True`, disadvantage if `False`, or normally if `None`.

Returns A SimpleRollResult describing the rolled save.

Return type *SimpleRollResult*

property saves
The saves of the creature.

Return type *AliasSaves*

set_ac(*ac: int*)
Sets the combatant's armor class.

Parameters **ac** (*int*) – The new AC.

set_group(*group*)
Sets the combatant's group

Parameters **group** (*str*) – The name of the group. None to remove from group.

Returns The combatant's new group, or None if the combatant was removed from a group.

Return type *SimpleGroup* or None

set_hp(*new_hp*)
Sets the creature's remaining HP.

Parameters **new_hp** (*int*) – The amount of remaining HP (a nonnegative integer).

set_init(*init: int*)
Sets the combatant's initiative roll.

Parameters **init** (*int*) – The new initiative.

set_maxhp(*maxhp: int*)
Sets the combatant's max HP.

Parameters **maxhp** (*int*) – The new max HP.

set_name(*name: str*)
Sets the combatant's name.

Parameters **name** (*str*) – The new name.

set_note(*note: str*)
Sets the combatant's note.

Parameters **note** (*str*) – The new note.

set_temp_hp(*new_temp*)
Sets a creature's temp HP.

Parameters **new_temp** (*int*) – The new temp HP (a non-negative integer).

set_thp(*thp: int*)
Deprecated since version 2.5.0: Use `SimpleCombatant.set_temp_hp()` instead.

Sets the combatant's temp HP.

Parameters **thp** (*int*) – The new temp HP.

property skills
The skills of the creature.

Return type *AliasSkills*

property spellbook
The creature's spellcasting information.

Return type *AliasSpellbook*

property stats
The stats of the creature.

Return type *AliasBaseStats*

property temp_hp
The current temp HP of the creature.

Return type int

property temphp
Deprecated since version 2.5.0: Use `SimpleCombatant.temp_hp` instead.

How many temporary hit points the combatant has.

Return type int

wouldhit(*to_hit: int*)
Deprecated since version 1.1.5: Use `to_hit >= combatant.ac` instead.

Checks if a roll would hit this combatant.

Parameters **to_hit** (*int*) – The rolled total.

Returns Whether the total would hit.

Return type bool

7.7.3 SimpleGroup

class SimpleGroup

combatants
A list of all *SimpleCombatant* in this group.

Type list of *SimpleCombatant*

type
The type of the object (`"group"`), to determine whether this is a group or not.

Type str

get_combatant(*name*)
Gets a *SimpleCombatant*, fuzzy searching (partial match) on name.

Parameters **name** (*str*) – The name of the combatant to get.

Returns The combatant.

Return type *SimpleCombatant*

property name
The name of the group.

Return type str

7.7.4 SimpleEffect

class SimpleEffect

combatant_name
The name of the combatant this effect is on.

Type str

conc
Whether the effect requires concentration.

Type bool

desc
The description of the effect.

Type str

duration
The initial duration of the effect, in rounds (`-1` = infinite).

Type int

effect
The applied effect of the object.

Type dict

name
The name of the effect.

Type str

remaining
The remaining duration of the effect, in rounds.

Type int

ticks_on_end
Whether the effect duration ticks at the end of the combatant's turn or at the start.

Type bool

property children
Gets the child effects of this effect.

Return type list of *SimpleEffect*

property parent
Gets the parent effect of this effect, or `None` if this effect has no parent.

Return type *SimpleEffect* or None

set_parent(*parent*)
Sets the parent effect of this effect.

Parameters **parent** (*SimpleEffect*) – The parent.

7.8 SimpleRollResult

class SimpleRollResult

dice
The rolled dice (e.g. `1d20 (5)`).

Type str

total
The total of the roll.

Type int

full
The string representing the roll.

Type str

result
The RollResult object returned by the roll.

Type `d20.RollResult`

raw
The Expression object returned by the roll. Equivalent to `SimpleRollResult.result.expr`.

Type `d20.Expression`

consolidated()
Gets the most simplified version of the roll string. Consolidates totals and damage types together.

Note that this modifies the result expression in place!

```
>>> result = vroll("3d6[fire]+1d4[cold]")
>>> str(result)
'3d6 (3, 3, 2) [fire] + 1d4 (2) [cold] = `10`'
>>> result.consolidated()
'8 [fire] + 2 [cold]'
```

Return type str

__str__()
Equivalent to `result.full`.

7.9 ParsedArguments

class ParsedArguments

add_context(*context*, *args*)
Adds contextual parsed arguments (arguments that only apply in a given context)

Parameters

- **context** – The context to add arguments to.
- **args** (*ParsedArguments*) – The arguments to add.

adv(*ea=False*, *boolwise=False*, *ephem=False*, *custom: dict = None*)
Determines whether to roll with advantage, disadvantage, Elven Accuracy, or no special effect.

Parameters

- **ea** – Whether to parse for elven accuracy.
- **boolwise** – Whether to return an integer or tribool representation.
- **ephem** – Whether to return an ephemeral argument if such exists.
- **custom** – Dictionary of custom values to parse for. There should be a key for each value you want to overwrite. `custom={'adv': 'custom_adv'}` would allow you to parse for advantage if the `custom_adv` argument is found.

Returns -1 for dis, 0 for normal, 1 for adv, 2 for ea

get(*arg*, *default=None*, *type_=<class 'str'>*, *ephem=False*)
Gets a list of all values of an argument.

Parameters

- **arg** (*str*) – The name of the arg to get.
- **default** – The default value to return if the arg is not found. Not cast to type.
- **type** – The type that each value in the list should be returned as.
- **ephem** (*bool*) – Whether to add applicable ephemeral arguments to the returned list.

Returns The relevant argument list.

Return type list

ignore(*arg*)
Removes any instances of an argument from the result in all contexts (ephemeral included).

Parameters **arg** – The argument to ignore.

join(*arg*, *connector: str*, *default=None*, *ephem=False*)
Returns a str formed from all of one arg, joined by a connector.

Parameters

- **arg** – The arg to join.
- **connector** – What to join the arg by.
- **default** – What to return if the arg does not exist.
- **ephem** – Whether to return an ephemeral argument if such exists.

Returns The joined str, or default.

last(*arg*, *default=None*, *type_: type = <class 'str'>*, *ephem=False*)
Gets the last value of an arg.

Parameters

- **arg** (*str*) – The name of the arg to get.
- **default** – The default value to return if the arg is not found. Not cast to type.
- **type** – The type that the arg should be returned as.
- **ephem** – Whether to return an ephemeral argument if such exists.

Raises InvalidArgument if the arg cannot be cast to the type

Returns The relevant argument.

set_context(*context*)
Sets the current argument parsing context.

Parameters **context** – Any hashable context.

update(*new*)
Updates the arguments in this argument list from a dict.

Parameters **new** (*dict[str, str] or dict[str, list[str]]*) – The new values for each argument.

update_nx(*new*)
Like `.update()`, but only fills in arguments that were not already parsed. Ignores the argument if the value is None.

Parameters **new** (*dict[str, str] or dict[str, list[str]] or dict[str, None]*) – The new values for each argument.

7.10 Context Models

7.10.1 AliasContext

class AliasContext
Used to expose some information about the context, like the guild name, channel name, author name, and current prefix to alias authors.

You can access this in an alias by using the `ctx` local.

property alias
The name the alias was invoked with. Note: When used in a base command, this will return the deepest sub-command, but when used in an alias it will return the base command.

```
>>> !test {{ctx.alias}}
'test'
```

Return type str

property author
The user that ran the alias.

Return type *AliasAuthor*

property channel
The channel the alias was run in.

Return type *AliasChannel*

property guild
The discord guild (server) the alias was run in, or None if the alias was run in DMs.

Return type *AliasGuild* or None

property prefix
The prefix used to run the alias.

Return type str

7.10.2 AliasGuild

class AliasGuild
Represents the Discord guild (server) an alias was invoked in.

property id
The ID of the guild.

Return type int

property name
The name of the guild.

Return type str

7.10.3 AliasChannel

class AliasChannel
Represents the Discord channel an alias was invoked in.

property category
The category of the channel the alias was run in

Return type *AliasCategory* or None

property id
The ID of the channel.

Return type int

property name
The name of the channel, not including the preceding hash (#).

Return type str

property topic
The channel topic.

Return type str

7.10.4 AliasCategory

class AliasCategory
Represents the category of the Discord channel an alias was invoked in.

property id
The ID of the category.

Return type int

property name
The name of the category

Return type str

7.10.5 AliasAuthor

class AliasAuthor
Represents the Discord user who invoked an alias.

property discriminator
The user's discriminator (number after the hash).

Return type str

property display_name
The user's display name - nickname if applicable, otherwise same as their name.

Return type str

property id
The user's ID.

Return type int

property name
The user's username (not including the discriminator).

Return type str

7.11 AliasCharacter

class AliasCharacter(*AliasStatBlock*)

property ac
The armor class of the creature.

Return type int or None

property attacks
The attacks of the creature.

Return type *AliasAttackList*

property background
Gets the character's background.

Return type str or None

cc(*name*)
Gets the AliasCustomCounter with the name *name*

Parameters **name** (*str*) – The name of the custom counter to get.

Returns The custom counter.

Return type *AliasCustomCounter*

Raises `ConsumableException` if the counter does not exist.

cc_exists(*name*)
Returns whether a custom counter exists.

Parameters **name** (*str*) – The name of the custom counter to check.

Returns Whether the counter exists.

cc_str(*name*)
Returns a string representing a custom counter.

Parameters **name** (*str*) – The name of the custom counter to get.

Returns A string representing the current value, maximum, and minimum of the counter.

Return type str

Raises `ConsumableException` if the counter does not exist.

Example:

```
>>> cc_str("Ki")
'11/17'
>>> cc_str("Bardic Inspiration")
''
```

property consumables

Returns a list of custom counters on the character.

Return type list[*AliasCustomCounter*]

create_cc(*name: str*, **args*, ***kwargs*)

Creates a custom counter. If a counter with the same name already exists, it will replace it.

Parameters

- **name** (*str*) – The name of the counter to create.
- **minVal** (*str*) – The minimum value of the counter. Supports *Cvar Table* parsing.
- **maxVal** (*str*) – The maximum value of the counter. Supports *Cvar Table* parsing.
- **reset** (*str*) – One of `'short'`, `'long'`, `'hp'`, `'none'`, or `None`.
- **dispType** (*str*) – Either `None` or `'bubble'`.
- **reset_to** (*str*) – The value the counter should reset to. Supports *Cvar Table* parsing.
- **reset_by** (*str*) – How much the counter should change by on a reset. Supports dice but not cvars.
- **title** (*str*) – The title of the counter.
- **desc** (*str*) – The description of the counter.

Return type *AliasCustomCounter*

Returns The newly created counter.

create_cc_nx(*name: str*, *minVal: str = None*, *maxVal: str = None*, *reset: str = None*, *dispType: str = None*, *reset_to: str = None*, *reset_by: str = None*, *title: str = None*, *desc: str = None*)

Creates a custom counter if one with the given name does not already exist. Equivalent to:

```
>>> if not cc_exists(name):
>>>     create_cc(name, minVal, maxVal, reset, dispType, reset_to, reset_by,␣
↪title, desc)
```

property csettings

Gets a copy of the character's settings dict.

Return type dict

property cvars

Returns a dict of cvars bound on this character.

Return type dict

property death_saves

Returns the characters death saves.

Return type *AliasDeathSaves*

delete_cc(*name*)
Deletes a custom counter.

Parameters **name** (*str*) – The name of the custom counter to delete.

Raises `ConsumableException` if the counter does not exist.

delete_cvar(*name*)
Deletes a custom character variable. Does nothing if the cvar does not exist.

Parameters **name** (*str*) – The name of the variable to delete.

get_cc(*name*)
Gets the value of a custom counter.

Parameters **name** (*str*) – The name of the custom counter to get.

Returns The current value of the counter.

Return type int

Raises `ConsumableException` if the counter does not exist.

get_cc_max(*name*)
Gets the maximum value of a custom counter.

Parameters **name** (*str*) – The name of the custom counter maximum to get.

Returns The maximum value of the counter. If a counter has no maximum, it will return INT_MAX (2^31-1).

Return type int

Raises `ConsumableException` if the counter does not exist.

get_cc_min(*name*)
Gets the minimum value of a custom counter.

Parameters **name** (*str*) – The name of the custom counter minimum to get.

Returns The minimum value of the counter. If a counter has no minimum, it will return INT_MIN (-2^31).

Return type int

Raises `ConsumableException` if the counter does not exist.

property hp
The current HP of the creature.

Return type int or None

hp_str()
Returns a string describing the creature's current, max, and temp HP.

Return type str

property levels
The levels of the creature.

Return type *AliasLevels*

property max_hp
The maximum HP of the creature.

Return type int or None

mod_cc(*name*, *val: int*, *strict=False*)
Modifies the value of a custom counter. Equivalent to `set_cc(name, get_cc(name) + value, strict)`.

modify_hp(*amount*, *ignore_temp=False*, *overflow=True*)
Modifies the creature's remaining HP by a given amount.

Parameters

- **amount** (*int*) – The amount of HP to add/remove.
- **ignore_temp** (*bool*) – If *amount* is negative, whether to damage temp HP first or ignore temp.
- **overflow** (*bool*) – If *amount* is positive, whether to allow overhealing or cap at the creature's max HP.

property name
The name of the creature.

Return type str

property owner
Returns the id of this character's owner.

Return type int

property race
Gets the character's race.

Return type str or None

reset_hp()
Heals a creature to max and removes any temp HP.

property resistances
The resistances, immunities, and vulnerabilities of the creature.

Return type *AliasResistances*

property saves
The saves of the creature.

Return type *AliasSaves*

set_cc(*name*, *value: int*, *strict=False*)
Sets the value of a custom counter.

Parameters

- **name** (*str*) – The name of the custom counter to set.
- **value** (*int*) – The value to set the counter to.
- **strict** (*bool*) – If `True`, will raise a `CounterOutOfBounds` if the new value is out of bounds, otherwise silently clips to bounds.

Raises `ConsumableException` if the counter does not exist.

Returns The cc's new value.

Return type int

set_cvar(*name*, *val: str*)
Sets a custom character variable, which will be available in all scripting contexts using this character.

Parameters

- **name** (*str*) – The name of the variable to set. Must be a valid identifier and not be in the *Cvar Table*.
- **value** (*str*) – The value to set it to.

set_cvar_nx(*name*, *val: str*)
Sets a custom character variable if it is not already set.

Parameters

- **name** (*str*) – The name of the variable to set. Must be a valid identifier and not be in the *Cvar Table*.
- **value** (*str*) – The value to set it to.

set_hp(*new_hp*)
Sets the creature's remaining HP.

Parameters **new_hp** (*int*) – The amount of remaining HP (a nonnegative integer).

set_temp_hp(*new_temp*)
Sets a creature's temp HP.

Parameters **new_temp** (*int*) – The new temp HP (a non-negative integer).

property sheet_type
Returns the sheet type of this character (beyond, dicecloud, google).

Return type str

property skills
The skills of the creature.

Return type *AliasSkills*

property spellbook
The creature's spellcasting information.

Return type *AliasSpellbook*

property stats
The stats of the creature.

Return type *AliasBaseStats*

property temp_hp
The current temp HP of the creature.

Return type int

property upstream
Returns the upstream key for this character.

Return type str

7.11.1 AliasCustomCounter

class AliasCustomCounter

property desc
Returns the cc's description.

Return type str or None

property display_type
Returns the cc's display type. (None, 'bubble')

Return type str

property max
Returns the maximum value of the cc, or 2^31-1 if the cc has no max.

Return type int

property min
Returns the minimum value of the cc, or -2^31 if the cc has no min.

Return type int

property name
Returns the cc's name.

Return type str

reset()
Resets the cc to its reset value. Errors if the cc has no reset value or no reset.

The reset value is calculated in 3 steps: - if the cc has a `reset_to` value, it is reset to that - else if the cc has a `reset_by` value, it is modified by that much - else the reset value is its max

Return CustomCounterResetResult (new_value: int, old_value: int, target_value: int, delta: str)

property reset_by
Returns the amount the cc changes by on a reset, if it was created with an explicit `resetby`.

Returns The amount the cc changes by. Guaranteed to be a rollable string.

Return type str or None

property reset_on
Returns the condition on which the cc resets. ('long', 'short', 'none', None)

Return type str or None

property reset_to
Returns the value the cc resets to, if it was created with an explicit `resetto`.

Return type int or None

set(*new_value*, *strict=False*)
Sets the cc's value to a new value.

Parameters

- **new_value** (*int*) – The new value to set.
- **strict** (*bool*) – Whether to error when going out of bounds (true) or to clip silently (false).

Returns The cc's new value.

Return type int

property title
Returns the cc's title.

Return type str or None

property value
Returns the current value of the cc.

Return type int

7.11.2 AliasDeathSaves

class AliasDeathSaves

fail(*num=1*)
Adds one or more failed death saves.

Parameters **num** (*int*) – The number of failed death saves to add.

property fails
Returns the number of failed death saves.

Return type int

is_dead()
Returns whether or not the character is dead.

Return type bool

is_stable()
Returns whether or not the character is stable.

Return type bool

reset()
Resets all death saves.

succeed(*num=1*)
Adds one or more successful death saves.

Parameters **num** (*int*) – The number of successful death saves to add.

property successes
Returns the number of successful death saves.

Return type int

7.12 StatBlock Models

7.12.1 AliasStatBlock

class AliasStatBlock
A base class representing any creature (player or otherwise) that has stats.

Generally, these are never directly used - notable subclasses are *SimpleCombatant* and *AliasCharacter*.

property ac
The armor class of the creature.

Return type int or None

property attacks
The attacks of the creature.

Return type *AliasAttackList*

property hp
The current HP of the creature.

Return type int or None

hp_str()
Returns a string describing the creature's current, max, and temp HP.

Return type str

property levels
The levels of the creature.

Return type *AliasLevels*

property max_hp
The maximum HP of the creature.

Return type int or None

modify_hp(*amount*, *ignore_temp=False*, *overflow=True*)
Modifies the creature's remaining HP by a given amount.

Parameters

- **amount** (*int*) – The amount of HP to add/remove.
- **ignore_temp** (*bool*) – If *amount* is negative, whether to damage temp HP first or ignore temp.
- **overflow** (*bool*) – If *amount* is positive, whether to allow overhealing or cap at the creature's max HP.

property name
The name of the creature.

Return type str

reset_hp()
Heals a creature to max and removes any temp HP.

property resistances
The resistances, immunities, and vulnerabilities of the creature.

Return type *AliasResistances*

property saves
The saves of the creature.

Return type *AliasSaves*

set_hp(*new_hp*)
Sets the creature's remaining HP.

Parameters **new_hp** (*int*) – The amount of remaining HP (a nonnegative integer).

set_temp_hp(*new_temp*)
Sets a creature's temp HP.

Parameters **new_temp** (*int*) – The new temp HP (a non-negative integer).

property skills
The skills of the creature.

Return type *AliasSkills*

property spellbook
The creature's spellcasting information.

Return type *AliasSpellbook*

property stats
The stats of the creature.

Return type *AliasBaseStats*

property temp_hp
The current temp HP of the creature.

Return type int

7.12.2 AliasBaseStats

class AliasBaseStats
Represents a statblock's 6 base ability scores and proficiency bonus.

property charisma
Charisma score.

Return type int

property constitution
Constitution score.

Return type int

property dexterity
Dexterity score.

Return type int

get_mod(*stat: str*)
Gets the modifier for a base stat (str, dex, con, etc). Does *not* take skill check bonuses into account.

For the skill check modifier, use `StatBlock.skills.strength` etc.

Parameters **stat** (*str*) – The stat to get the modifier for.

Return type int

property intelligence
Intelligence score.

Return type int

property prof_bonus
The proficiency bonus.

Return type int

property strength
Strength score.

Return type int

property wisdom
Wisdom score.

Return type int

7.12.3 AliasLevels

class AliasLevels
Represents a statblock's class levels.

for (cls, level) in AliasLevels:
Iterates over pairs of class names and the number of levels in that class.

Type Iterable[tuple[str, int]]

get (*cls_name*, *default=0*)
Gets the levels in a given class, or *default* if there are none.

Parameters

- **cls_name** (*str*) – The name of the class to get the levels of.
- **default** (*int*) – What to return if the statblock does not have levels in the given class.

Return type float or int

property total_level
The total level.

Return type float or int

7.12.4 AliasAttackList

class AliasAttackList
A container of a statblock's attacks.

str (*AliasAttackList*)
Returns a string representation of all attacks in this attack list.

Type str

len (*AliasAttackList*)
Returns the number of attacks in this attack list.

Type int

for attack in AliasAttackList:
Iterates over attacks in this attack list.

Type Iterable[*AliasAttack*]

AliasAttackList[i]
Gets the *i*-th indexed attack.

Type *AliasAttack*

7.12.5 AliasAttack

class AliasAttack
An attack.

str (*AliasAttack*)
Returns a string representation of this attack.

Type str

property name
The name of the attack.

Return type str

property proper
Whether or not this attack is a proper noun.

Return type bool

property raw
A dict representing the raw value of this attack.

Return type dict

property verb
The custom verb used for this attack, if applicable.

Return type str or None

7.12.6 AliasSkill

class AliasSkill
A skill modifier.

property adv
The guaranteed advantage or disadvantage on this skill modifier. True = adv, False = dis, None = normal.

Return type bool or None

property bonus
The miscellaneous bonus to the skill modifier.

Return type int

d20(*base_adv=None*, *reroll=None*, *min_val=None*, *mod_override=None*)
Gets a dice string representing the roll for this skill.

Parameters

- **base_adv** (*bool*) – Whether this roll should be made at adv (True), dis (False), or normally (None).
- **reroll** (*int*) – If the roll lands on this number, reroll it once (Halfling Luck).
- **min_val** (*int*) – The minimum value of the dice roll (Reliable Talent, Glibness).
- **mod_override** (*int*) – Overrides the skill modifier.

Return type str

property prof
The proficiency multiplier in this skill. 0 = no proficiency, 0.5 = JoAT, 1 = proficiency, 2 = expertise.

Return type float or int

property value
The final modifier. Generally, `value = (base stat mod) + (profBonus) * prof + bonus`.

Return type int

7.12.7 AliasSkills

class AliasSkills
An object holding the skill modifiers for all skills.

for (skill_name, skill) in AliasSkills:
Iterates over pairs of skill names and corresponding skills.

Type Iterable[tuple[str, *AliasSkill*]]

acrobatics
animalHandling
arcana
athletics
deception
history
initiative
insight
intimidation
investigation
medicine
nature
perception
performance
persuasion
religion
sleightOfHand
stealth
survival
strength
dexterity
constitution
intelligence
wisdom
charisma
The skill modifier for the given skill.

Type *AliasSkill*

7.12.8 AliasSaves

class AliasSaves
An objecting holding the modifiers of all saves.

for (save_name, skill) in AliasSaves:
Iterates over pairs of save names and corresponding save.

Type Iterable[tuple[str, *AliasSkill*]]

get(*base_stat*)
Gets the save skill for a given stat (str, dex, etc).

Parameters **base_stat** (*str*) – The stat to get the save for.

Return type *AliasSkill*

7.12.9 AliasResistances

class AliasResistances
A statblock's resistances, immunities, vulnerabilities, and explicit neural damage types.

property immune
A list of damage types that the stat block is immune to.

Return type list[*Resistance*]

property neutral
A list of damage types that the stat block ignores in damage calculations. (i.e. will not handle resistances/vulnerabilities/immunities)

Return type list[*Resistance*]

property resist
A list of damage types that the stat block is resistant to.

Return type list[*Resistance*]

property vuln
A list of damage types that the stat block is vulnerable to.

Return type list[*Resistance*]

7.12.10 Resistance

class Resistance
Represents a conditional resistance to a damage type.

Only applied to a type token set T if $dtype \in T \land \neg(unless \cap T) \land only \subset T$.

Note: transforms all damage types given to lowercase.

dtype
The damage type.

Type str

unless
A set of tokens that if present, this resistance will not apply.

Type set[str]

only
A set of tokens that unless present, this resistance will not apply.

Type set[str]

applies_to(*tokens*)
Note that tokens should be a set of lowercase strings.

Parameters **tokens** (*set[str]*) – A set of strings to test against.

Return type bool

applies_to_str(*dtype*)
Returns whether or not this resistance is applicable to a damage type.

Parameters **dtype** (*str*) – The damage type to test.

Return type bool

7.12.11 AliasSpellbook

class AliasSpellbook
A statblock's spellcasting information.

spell in AliasSpellbook
Returns whether the spell named *spell* (str) is known.

Type bool

can_cast(*spell*, *level*)
Returns whether or not the given spell can currently be cast at the given level.

Parameters

- **spell** (*str*) – The name of the spell.
- **level** (*int*) – The level the spell is being cast at.

Return type bool

cast(*spell*, *level*)
Uses all resources to cast a given spell at a given level.

Parameters

- **spell** (*str*) – The name of the spell.
- **level** (*int*) – The level the spell is being cast at.

property caster_level
The caster's caster level.

Return type int

property dc
The spellcasting DC.

Return type int

get_max_slots(*level*)
Gets the maximum number of level *level* spell slots available.

Parameters **level** (*int*) – The spell level [1..9].

Returns int The maximum number of spell slots.

get_slots(*level*)
Gets the remaining number of slots of a given level. Always returns 1 if level is 0.

Parameters **level** (*int*) – The spell level to get the remaining slots of.

Returns int The number of slots remaining.

remaining_casts_of(*spell*, *level*)
Gets a string representing the remaining casts of a given spell at a given level.

Parameters

- **spell** (*str*) – The name of the spell (case-sensitive).
- **level** (*int*) – The level the spell is being cast at.

Return type str

reset_slots()
Resets the number of remaining spell slots of all levels to the max.

property sab
The spell attack bonus.

Return type int

set_slots(*level*, *value*)
Sets the remaining number of spell slots of a given level.

Parameters

- **level** (*int*) – The spell level to set [1..9].
- **value** (*int*) – The remaining number of slots.

slots_str(*level*)

Parameters **level** (*int*) – The level of spell slot to return.

Returns str A string representing the caster's remaining spell slots.

property spell_mod
The spellcasting modifier.

Return type int

property spells
The list of spells in this spellbook.

Return type list[*AliasSpellbookSpell*]

use_slot(*level*)
Uses one spell slot of a given level. Equivalent to `set_slots(level, get_slots(level) - 1)`.

Parameters **level** (*int*) – The level of spell slot to use.

AliasSpellbookSpell

class AliasSpellbookSpell

property dc
The spell's overridden DC. None if this spell uses the default caster DC.

Return type int or None

property mod
The spell's overridden spellcasting modifier. None if this spell uses the default caster spellcasting modifier.

Return type int or None

property name
The name of the spell.

Return type str

property sab
The spell's overridden spell attack bonus. None if this spell uses the default caster spell attack bonus.

Return type int or None

CHAPTER

EIGHT

AUTOMATION REFERENCE

This page details the structure of Avrae's Automation system, the backbone behind custom spells and attacks.

8.1 Basic Structure

An automation run is made up of a list of *effects*. See below for what each effect does.

All Automation runs provide the following variables:

- `caster` (*AliasStatBlock*) The character, combatant, or monster who is running the automation.
- `targets` (list of *AliasStatBlock*, *str*, or None) A list of combatants targeted by this automation (i.e. the `-t` argument).

8.2 Target

```
{
    type: "target";
    target: "all"|"each"|int|"self";
    effects: Effect[];
}
```

A Target effect should only show up as a top-level effect. It designates what creatures to affect.

`target`

- `"all"`: Affects all targets (usually save spells)
- `"each"`: Affects each target (usually attack spells)
- `int`: Affects the Nth target (1-indexed)
- `"self"`: Affects the caster

`effects`
A list of effects that each targeted creature will be subject to.

Variables

- `target` (*AliasStatBlock*) The current target.
- `targetIteration` (*int*) If running multiple iterations (i.e. `-rr`), the current iteration (1-indexed).

8.3 Attack

```
{
    type: "attack";
    hit: Effect[];
    miss: Effect[];
    attackBonus?: IntExpression;
}
```

An Attack effect makes an attack roll against a targeted creature. It must be inside a Target effect.

hit
: A list of effects to execute on a hit.

miss
: A list of effects to execute on a miss.

attackBonus
: *optional* - An IntExpression that details what attack bonus to use (defaults to caster's spell attack mod).

Variables

- `lastAttackDidHit` (`bool`) Whether the attack hit.
- `lastAttackDidCrit` (`bool`) If the attack hit, whether it crit.

8.4 Save

```
{
    type: "save";
    stat: "str"|"dex"|"con"|"int"|"wis"|"cha";
    fail: Effect[];
    success: Effect[];
    dc?: IntExpression;
}
```

A Save effect forces a targeted creature to make a saving throw. It must be inside a Target effect.

stat
: The type of saving throw.

fail
: A list of effects to execute on a failed save.

success
: A list of effects to execute on a successful save.

dc
: *optional* - An IntExpression that details what DC to use (defaults to caster's spell DC).

Variables

- `lastSaveDidPass` (`bool`) Whether the target passed the save.

8.5 Damage

```
{
    type: "damage";
    damage: AnnotatedString;
    overheal?: boolean;
    higher?: {int: string};
    cantripScale?: boolean;
}
```

Deals damage to a targeted creature. It must be inside a Target effect.

damage
: How much damage to deal. Can use variables defined in a Meta tag.

overheal
: New in version 1.4.1: *optional* - Whether this damage should allow a target to exceed its hit point maximum.

higher
: *optional* - How much to add to the damage when a spell is cast at a certain level.

cantripScale
: *optional* - Whether this roll should scale like a cantrip.

Variables

- `lastDamage` (*int*) The amount of damage dealt.

8.6 TempHP

```
{
    type: "temphp";
    amount: AnnotatedString;
    higher?: {int: string};
    cantripScale?: boolean;
}
```

Sets the target's THP. It must be inside a Target effect.

amount
: How much temp HP the target should have. Can use variables defined in a Meta tag.

higher
: *optional* - How much to add to the THP when a spell is cast at a certain level.

cantripScale
: *optional* - Whether this roll should scale like a cantrip.

Variables

- `lastTempHp` (*int*) The amount of temp HP granted.

8.7 IEffect

```
{
    type: "ieffect";
    name: string;
    duration: int | IntExpression;
    effects: AnnotatedString;
    end?: boolean;
    conc?: boolean;
    desc?: AnnotatedString;
}
```

Adds an InitTracker Effect to a targeted creature, if the automation target is in combat. It must be inside a Target effect.

name
The name of the effect to add.

duration
The duration of the effect, in rounds of combat. Can use variables defined in a Meta tag.

effects
The effects to add (see `add_effect()`). Can use variables defined in a Meta tag.

end
optional - Whether the effect timer should tick on the end of the turn, rather than start.

conc
optional - Whether the effect requires concentration.

desc
optional - The description of the effect (displays on combatant's turn).

8.8 Roll

```
{
    type: "roll";
    dice: AnnotatedString;
    name: string;
    higher?: {int: string};
    cantripScale?: boolean;
    hidden?: boolean;
}
```

Rolls some dice and saves the result in a variable. Displays the roll and its name in a Meta field, unless `hidden` is `true`.

dice
An AnnotatedString detailing what dice to roll.

name
What to save the result as.

higher
optional - How much to add to the roll when a spell is cast at a certain level.

cantripScale
optional - Whether this roll should scale like a cantrip.

hidden
optional - If `true`, won't display the roll in the Meta field, or apply any bonuses from -d.

Variables

- `lastRoll` (*int*) The total of the roll.

8.9 Text

```
{
    type: "text";
    text: AnnotatedString;
}
```

Outputs a short amount of text in the resulting embed.

text
An AnnotatedString detailing the text to display.

8.10 Set Variable

New in version 2.7.0.

```
{
    type: "variable";
    name: string;
    value: IntExpression;
    higher?: {int: IntExpression};
    onError?: IntExpression;
}
```

Saves the result of an `IntExpression` to a variable without displaying anything.

name
The name of the variable to save.

value
The value to set the variable to.

higher
optional - What to set the variable to instead when a spell is cast at a higher level.

onError
optional - If provided, what to set the variable to if the normal value would throw an error.

8.11 Condition

New in version 2.7.0.

```
{
    type: "condition";
    condition: IntExpression;
    onTrue: Effect[];
```

(continues on next page)

(continued from previous page)

```
    onFalse: Effect[];
    errorBehaviour?: "true" | "false" | "both" | "neither" | "raise";
}
```

Run certain effects if a special condition is met, or other effects otherwise.

condition
: The condition to check.

onTrue
: The effects to run if `condition` is `True` or any non-zero value.

onFalse
: The effects to run if `condition` is `False` or `0`.

errorBehaviour

: How to behave if the condition raises an error:

- `"true"`: Run the `onTrue` effects.
- `"false"`: Run the `onFalse` effects. (*default*)
- `"both"`: Run both the `onTrue` and `onFalse` effects, in that order.
- `"neither"`: Skip this effect.
- `"raise"`: Raise the error and halt execution.

8.12 AnnotatedString

An AnnotatedString is a string that can access saved variables. To access a variable, surround the name in brackets (e.g. `{damage}`). Available variables include:

- implicit variables from Effects (see relevant effect for a list of variables it provides)
- any defined in a `Roll` or `Set Variable` effect
- all variables from the *Cvar Table*

This will replace the bracketed portion with the value of the meta variable.

To perform math inside an AnnotatedString, surround the formula with two curly braces (e.g. `{{floor(dexterityMod+spell)}}`).

8.13 IntExpression

An IntExpression is similar to an AnnotatedString in its ability to use variables and functions. However, it has the following differences:

- Curly braces around the expression are not required
- An IntExpression can only contain one expression
- The result of an IntExpression must be an integer.

These are valid IntExpressions:

- `8 + proficiencyBonus + dexterityMod`

- `12`
- `floor(level / 2)`

These are *not* valid IntExpressions:

- `1d8`
- `DC {8 + proficiencyBonus + dexterityMod}`

8.14 Examples

8.14.1 Attack

A normal attack:

```
[
  {
    "type": "target",
    "target": "each",
    "effects": [
      {
        "type": "attack",
        "attackBonus": "dexterityMod + proficiencyBonus",
        "hit": [
          {
            "type": "damage",
            "damage": "1d10[piercing]"
          }
        ],
        "miss": []
      }
    ]
  }
]
```

8.14.2 Save

A spell that requires a Dexterity save for half damage:

```
[
  {
    "type": "roll",
    "dice": "8d6[fire]",
    "name": "damage",
    "higher": {
      "4": "1d6[fire]",
      "5": "2d6[fire]",
      "6": "3d6[fire]",
      "7": "4d6[fire]",
      "8": "5d6[fire]",
      "9": "6d6[fire]"
    }
  },
  {
```

(continues on next page)

(continued from previous page)

```
    "type": "target",
    "target": "all",
    "effects": [
      {
        "type": "save",
        "stat": "dex",
        "fail": [
          {
            "type": "damage",
            "damage": "{damage}"
          }
        ],
        "success": [
          {
            "type": "damage",
            "damage": "({damage})/2"
          }
        ]
      }
    ]
  },
  {
    "type": "text",
    "text": "Each creature in a 20-foot radius must make a Dexterity saving throw. A␣
↪target takes 8d6 fire damage on a failed save, or half as much damage on a␣
↪successful one."
  }
]
```

8.14.3 Attack & Save

An attack from a poisoned blade:

```
[
  {
    "type": "target",
    "target": "each",
    "effects": [
      {
        "type": "attack",
        "attackBonus": "strengthMod + proficiencyBonus",
        "hit": [
          {
            "type": "damage",
            "damage": "1d10[piercing]"
          },
          {
            "type": "save",
            "stat": "con",
            "dc": "12",
            "fail": [
              {
                "type": "damage",
                "damage": "1d6[poison]"
              }
```

(continues on next page)

(continued from previous page)

```
            ],
            "success": []
          }
        ],
        "miss": []
      }
    ]
  },
  {
    "type": "text",
    "text": "On a hit, a target must make a DC 12 Constitution saving throw or take␣
↪1d6 poison damage."
  }
]
```

8.14.4 Draining Attack

An attack that heals the caster for half the amount of damage dealt:

```
[
  {
    "type": "variable",
    "name": "lastDamage",
    "value": "0"
  },
  {
    "type": "target",
    "target": "each",
    "effects": [
      {
        "type": "attack",
        "attackBonus": "charismaMod + proficiencyBonus",
        "hit": [
          {
            "type": "damage",
            "damage": "3d6[necrotic]"
          }
        ],
        "miss": []
      }
    ]
  },
  {
    "type": "target",
    "target": "self",
    "effects": [
      {
        "type": "damage",
        "damage": "-{lastDamage}/2 [heal]"
      }
    ]
  },
  {
    "type": "text",
    "text": "On a hit, the target takes 3d6 necrotic damage, and you regain hit␣
↪points equal to half the amount of necrotic damage dealt."
```

(continues on next page)

(continued from previous page)

```
  }
]
```

8.14.5 Target Health-Based

A spell that does different amounts of damage based on whether or not the target is damaged:

```
[
  {
    "type": "target",
    "target": "each",
    "effects": [
      {
        "type": "save",
        "stat": "wis",
        "fail": [
          {
            "type": "condition",
            "condition": "target.hp < target.max_hp",
            "onTrue": [
              {
                "type": "damage",
                "damage": "1d8 [necrotic]"
              }
            ],
            "onFalse": [
              {
                "type": "damage",
                "damage": "1d4 [necrotic]"
              }
            ],
            "errorBehaviour": "both"
          }
        ],
        "success": []
      }
    ]
  },
  {
    "type": "text",
    "text": "The target must succeed on a Wisdom saving throw or take 1d4 necrotic␣
↪damage. If the target is missing any of its hit points, it instead takes 1d8␣
↪necrotic damage."
  }
]
```

8.14.6 Area Draining Effect

An effect that heals the caster for the total damage dealt:

```
[
  {
    "type": "variable",
```

(continues on next page)

(continued from previous page)

```
    "name": "totalDamage",
    "value": "0"
  },
  {
    "type": "target",
    "target": "each",
    "effects": [
      {
        "type": "damage",
        "damage": "1d6 [necrotic]"
      },
      {
        "type": "variable",
        "name": "totalDamage",
        "value": "totalDamage + lastDamage"
      }
    ]
  },
  {
    "type": "target",
    "target": "self",
    "effects": [
      {
        "type": "damage",
        "damage": "-{totalDamage} [heal]"
      }
    ]
  },
  {
    "type": "text",
    "text": "Each creature within 10 feet of you takes 1d6 necrotic damage. You␣
↪regain hit points equal to the sum of the necrotic damage dealt."
  }
]
```

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

A

B

C

D

E

F

G

H

S

T

U

V

W