avrae

Andrew Zhu, D&D Beyond

Jan 15, 2025

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 v1, Dicecloud v2, 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:

```
!import 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 v1

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

```
!import https://v1.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 Dicecloud v2

To add a character from Dicecloud V2, use the following command:

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

1.2.4 Google Sheets

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

```
!import 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
```

(continues on next page)

(continued from previous page)

```
!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

INLINE ROLLING

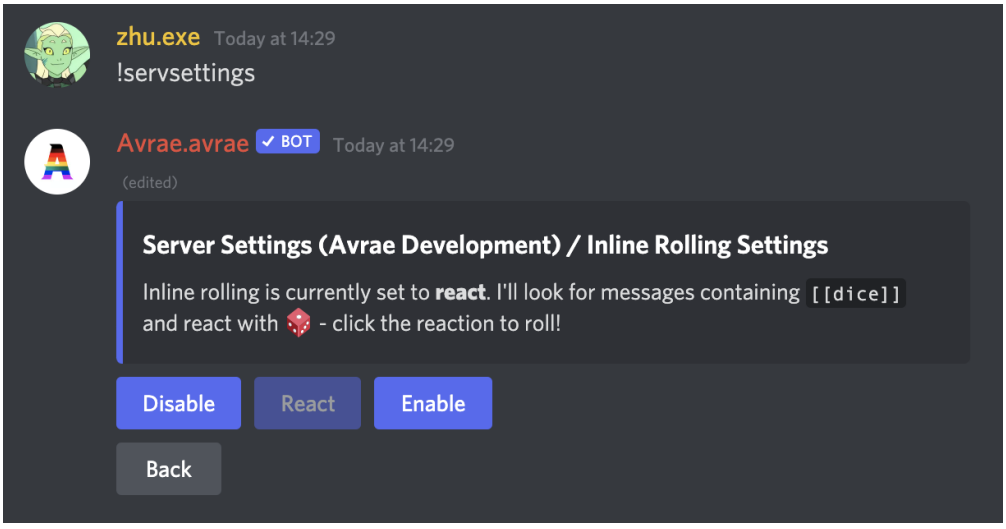
Whenever you send a message with some dice in between double brackets (e.g. `[[1d20]]`), Avrae will reply to it with a roll for each one. You can send messages with multiple, too, for example:

```
I attack the goblin with my shortsword [[1d20 + 6]] for a total of [[1d6 + 3]] piercing␣
↪damage.
```

4.1 Enabling Inline Rolling

By default, inline rolling is disabled when Avrae first joins a server. To enable Inline Rolling, a server admin (i.e. any member with the Manage Server permission) can enable it using the `!servsettings` command.

Select `Inline Rolling Settings` in the menu, and select whether to use reaction-based rolling or always-on rolling.



4.1.1 Always-On Rolling

When Avrae detects an inline roll in a message, she will immediately reply with the roll results for each roll in the message.

4.1.2 Reaction-Based Rolling

When Avrae detects an inline roll in a message, she will react to the message with the emoji. The message author can then react to have Avrae reply with the roll results for each roll in the message. Other users' reactions and any of the author's reactions beyond the first will be ignored.

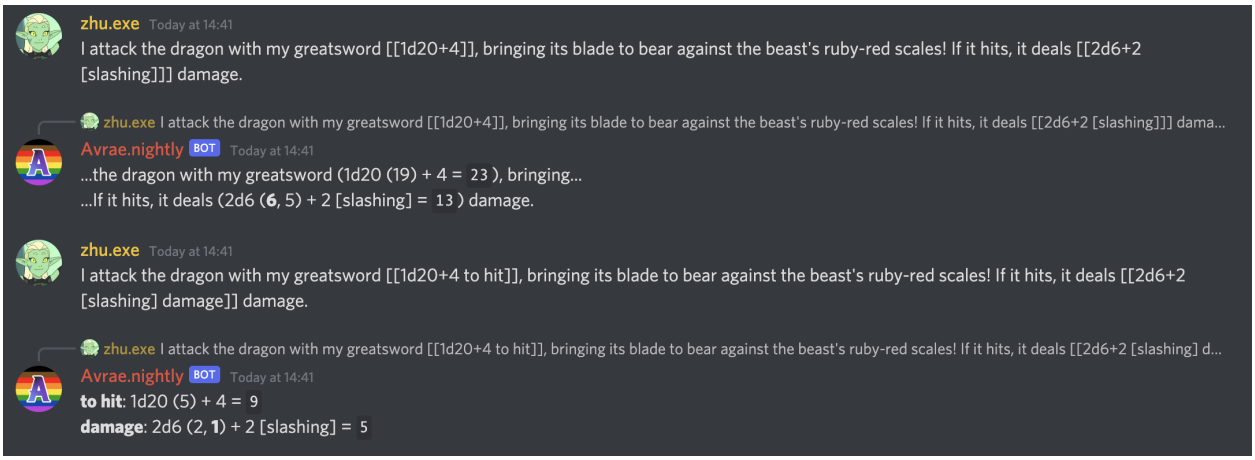
4.2 Arguments

Inline Rolling supports the `adv` and `dis` arguments in the same manner as the `!r` command.

4.3 Comments

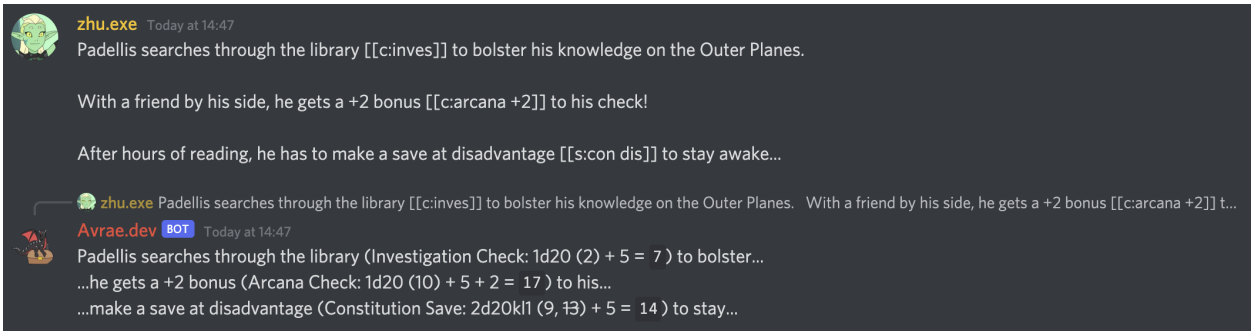
Inline Rolling supports comments similar to the `!r` command.

If a comment is supplied, the output will display the comment *instead* of the message context surrounding the roll.



4.4 Character Rolls

If an inline roll starts with `c:` or `s:`, the roll will use your active character's check or save dice for the given skill, respectively. This roll type can be combined with other dice bonuses or the advantage arguments.



4.5 Examples

Message	Description
`[[1d20]]`	Rolls 1d20.
`[[1d20+5 adv]]`	Rolls a d20 at advantage with a +5 bonus.

`[[4d6kh3 STR]]`
`[[4d6kh3 DEX]]`
`[[4d6kh3 CON]]`
`[[4d6kh3 INT]]`
`[[4d6kh3 WIS]]`
`[[4d6kh3 CHA]]`
	Rolls 6 sets of 4d6, keeping the highest 3 die of each set.

`[[c:arc]]`	Rolls an arcana check for the current active character.
`[[s:dex]]`	Rolls a dexterity save for the current active character.
`[[c:pers adv]]`	Rolls a persuasion check at advantage.
`[[s:con-2]]`	Rolls a constitution save with a -2 penalty.
`[[s:str+1d4 adv]]`	Rolls a strength save with a +1d4 bonus at advantage.

CHAPTER

FIVE

ADDITIONAL AUTOMATION SUPPORT

There are many spells and actions that have additional functionality that is not explained or detailed in the relevant lookup or `!help`. This page is meant to document those quirks and how to use them.

5.1 Specifying Class Feature DC Bonuses

You can grant bonuses to your class Saving Throw DCs by creating a cvar: `XDCBonus` (`WarlockDCBonus`, `BloodHunterDCBonus`, `MonkDCBonus`, etc).

This is to account for items such as the Dragonhide Belt, which adds a flat +1/2/3 bonus to the save DC for your class.

Note: `XDCBonus` is not generated by the the sheet itself, but is instead set by the user. It will grant the given bonus to the save DC of actions for that class. You can set it with `!cvar XDCBonus #`, such as `!cvar MonkDCBonus 2`, or with Draconic using `set_cvar()`.

This cvar should be an integer, or it could cause the automation to not run.

For more details on implementing this in homebrew, see *Specifying Class Feature DC Bonuses*

5.2 Spells with Additional Support

Note: `-choice` always transforms the argument to lowercase (`-choice FIre` will be treated as `-choice fire`), making it case insensitive.

Additionally, all of the options below use the *in* operator, so you can use partial matches (`-choice fire` will match `-choice fireball`)

5.2.1 Absorb Elements

You can have the automation automatically apply the damage type resistance you want to absorb using `-choice [element]` (`-choice fire` for instance).

Additionally, you can have the automation heal you for half the damage you absorb by adding `-amt [amount]` to the end of the command. (`-amt 5` for instance). It will half the damage you specify, and heal you for that amount.

You can use both of these together (`-choice fire -amt 5` for instance).

5.2.2 Alter Self

You can specify the type of alteration you want to take with the `-choice` argument.

- `-choice aquaticswimwater` for the Aquatic Adaptation option
- `-choice appearancealter` for the Change Appearance option
- `-choice naturalweapon` for the Natural Weapons option

5.2.3 Blindness/Deafness

You can specify the type of hindrance you want to apply with the `-choice` argument.

- `-choice blindness` for the Blindness option
- `-choice deafness` for the Deafness option

5.2.4 Dragon's Breath

You can specify the type of damage you want to apply with the `-choice` argument. If a choice is *not* provided at cast time, it will do [chromatic] damage by default, and you will need to use `-choice [element]` to specify the damage type each time the breath attack is used, and adjustments may be required to the targets health, depending on resistances.

In this case, it does not check against a value, but instead just inputs whatever is given. This means you can technically give someone Pizza Breath with `-choice pizza`.

5.2.5 Eldritch Blast

Eldritch Blast has a number of Eldritch Invocations that can affect it. You can add these to your `invocations` cvar to have them automatically applied when you cast Eldritch Blast.

If you add `agonizing blast` to your `invocations` cvar (`!cvar invocations agonizing blast`), it will automatically add your charisma when casting blast now.

Additionally, if you have other (official) invocations that affect blast, you can similarly add those to that cvar (`!cvar invocations agonizing blast, lance of lethargy` for instance), and it will add reminder text for them as well.

Like `-choice`, the automation will check against a lowercase version of the cvar, so `!cvar invocations Agonizing Blast` will work just as well as `!cvar invocations agonizing blast`.

Table 1: Supported Invocations

Invocation Name	Automation Support
Agonizing Blast	Adds your Charisma modifier to the damage
Repelling Blast	Includes reminder text
Eldritch Spear	Includes reminder text
Grasp Of Hadar	Includes reminder text
Lance Of Lethargy	Includes reminder text

5.2.6 Enhance Ability

You can specify the ability you want to enhance with the `-choice` argument.

- `-choice bearsconstitution`
- `-choice bullsstrength`
- `-choice catsdexterity`
- `-choice eaglescharisma`
- `-choice foxsintelligence`
- `-choice owlswisdom`

5.2.7 Enlarge/Reduce

You can specify the adjustment you want to apply with the `-choice` argument.

- `-choice enlarged` to enlarge the targets size
- `-choice reduced` to reduce the targets size

5.2.8 Eyebite

You can specify the type of effect you want to apply with the `-choice` argument.

- `-choice asleep` to apply the Asleep effect
- `-choice panicked` to apply the Panicked effect
- `-choice sickened` to apply the Sickened effect

5.2.9 Fire Shield

You can specify the type of shield you want to apply with the `-choice` argument.

- `-choice warmfire` to create a Warm Shield
- `-choice chillcold` to create a Chill Shield

5.2.10 Flame Strike

You can specify the type of additional damage you want to apply with the `-choice` argument when upcasting. If a choice is *not* provided at cast time, it will do [choice] damage by default, and adjustments may be required to the targets health, depending on resistances.

In this case, it does not check against a value, but instead just inputs whatever is given. This means you can technically have it be a Pizza Strike and deal extra [pizza] damage with `-choice pizza`.

5.2.11 Guardian of Nature

You can specify the type of form you want to apply with the `-choice` argument.

- `-choice "primal beast"` for the Primal Beast option
- `-choice "great tree"` for the Great Tree option

5.2.12 Hex

You can specify the affected ability score with `-choice`. This also applies to the `Shift Hex` action the spell grants for shifting the hex after the target dies.

- `-choice strength`
- `-choice dexterity`
- `-choice constitution`
- `-choice intelligence`
- `-choice wisdom`
- `-choice charisma`

5.2.13 Shield

You can have the automation automatically heal you for the damage you absorb by adding `-amt [amount]` to the end of the command. (`-amt 5` for instance). It will heal you for the amount you specify.

5.2.14 Spirit Guardians

You can specify the type of damage you want to apply with the `-choice` argument.

- `-choice goodneutralangelicfeyfae` for the Radiant damage option
- `-choice evilfiendish` for the Necrotic damage option

5.2.15 Spirit Shroud

You can specify the type of damage you want to apply to the granted attack with the `-choice` argument. If a choice is *not* provided at cast time, it will do [shroud] damage by default, and adjustments may be required to the targets health, depending on resistances.

In this case, it does not check against a value, but instead just inputs whatever is given. This means you can technically have it be a Pizza Shroud deal [pizza] damage with `-choice pizza`.

CHAPTER

SIX

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.

6.1 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:

6.2 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.

6.3 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.

6.4 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!

6.5 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

SEVEN

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.

7.1 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.

7.2 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".

7.3 Help

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

CHAPTER

EIGHT

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, @Ydomat#2886 for converting them to this format, and @mobius#1442 for updating them!

8.1 Half-Orc Relentless Endurance Tutorial

By @silverbass#2407, rewritten in Drac2 by @mobius#1442

```
!alias orc-relentless
```

This sets the alias name. If creating this alias in the Avrae workshop, you'll leave this line out.

```
embed
```

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

```
<drac2>
```

This specifies the start of a code block that will contain all the logic for the alias.

```
#Define variables for later use
cc = "Relentless Endurance"
desc = "When you are reduced to 0 hit points but not killed outright, you can drop to 1␣
↪hit point instead."
rest = "You can't use this feature again until you finish a long rest."
hasHP = "You have not been reduced to 0 hit points."
noCC = "You do not have this ability."
```

This defines some string variables that the alias will use in various places. Defining them as variables allows us to use the same strings in multiple places more easily, and makes the code more legible. This line: `#Define variables for later use` is a comment. Anything starting with a `#` is ignored when the alias runs, and can be used to make your alias code more readable and easier to follow

```
ch=character()
```

This alias will be accessing the active character several times, so this defines a variable to store it for easier access.

```
#Create the counter if it should exist but doesn't already
if ch.race.lower() == "half-orc":
    ch.create_cc_nx(cc, 0, 1, "long", "bubble", None, None, cc, desc+" "+rest)
```

The alias uses a custom counter to track the use of this ability. If the character was imported from Beyond, it should create the custom counter automatically. In case the character doesn't have the custom counter, for whatever reason, this code checks if the character's race is Half-Orc and creates it.

`if ch.race.lower() == "half-orc":`

This is a simple if-statement. We check if the character's race is Half-Orc. The `lower()` after the race makes it lower-case. We do this because string comparisons are case-sensitive, and making it all lower-case means we don't have to check for Half-Orc, Half-orc, and half-orc individually. Note the `:`. Forgetting it is a common error when using if blocks. The code inside the block will only execute if this condition is true.

`ch.create_cc_nx(cc, 0, 1, "long", "bubble", None, None, cc, desc+" "+rest)`

This code will run only if the if statement is true. That is, if the character's race is half-orc. Pay attention to the indentation shown in the code block above; this is another common error when writing if-blocks. Any code to be executed inside the block must be indented, and must all have the same indentation. Tabs or spaces will work, but you can't mix-and match them. Each line in the block must have have the same amount and type of leading white space.

So what does this line do? It has a lot of parts, so let's look at them in-order:

- `ch.create_cc_nx` This will create a custom counter on the character (`ch`) if it doesn't already exist.
- `cc` This defines the name of the counter. In this case, it uses one of the variables declared earlier, so the counter will be `Relentless Endurance`
- `0, 1` The next two arguments define the minimum and maximum values of the CC. Since this can only ever be used once at a time, this counter can only go between 0 (used) and 1 (available)
- `"long"` Next we define how the counter resets. We're specifying that it should reset on a Long Rest.
- `"bubble"` This specifies how the counter should be presented. Bubble gives a depiction of the counter that is more visual and aesthetically pleasing
- `None, None` These next two are Reset To and Reset By, respectively. They are optional arguments for more advanced custom counters, and aren't needed for this one.
- `cc` The next argumet is the Title of the counter that will be seen when setting or viewing the counter. We're just setting it to the same thing as its title.
- `desc+" "+rest` Finally, this is the counter's description. We're using two of the previously-defined variables, joined with a space between them.

```
#Logic of the alias. Check for all the necessary conditions
succ = "tries to use"
if ch.cc_exists(cc) and ch.get_cc(cc) and not ch.hp:
    succ = "uses"
    D = desc
    ch.mod_cc(cc, -1)
    ch.set_hp(1)
```

Another if-block, this one a little more complex than the last. We're checking more things here, and then executing more code if it meets all the conditions. Let's break it down.

`succ = "tries to use"` We're starting with this variable and giving it a default value. We'll change it later if the alias succeeds.

Taking a closer look at the if-statement:

`if ch.cc_exists(cc) and ch.get_cc(cc) and not ch.hp:`

This checks if all of the trigger conditions are valid. The `and` combining each statement means that all of the following conditions must be met.

- `ch.cc_exists(cc)` This checks if this character (`ch`) have a custom counter (`cc_exists`) called "Relentless Endurance" (`(cc)`)
- `ch.get_cc(cc)` This gets the value of the counter, which will be 0 (used) or 1 (not used). If-checks treat zero as False, and non-zero as True. So, if the counter is used, the if-check will fail here.
- `not ch.hp` Checks the character's hp. As before, zero hit points will be considered False, and non-zero is True. The `not` before hand will reverse that. That means that if the character has any HP left, the if-check will fail.

If all the conditions are met, the alias will execute the code inside the block. Note that each of these lines has the same indentation. This block will do most of the mechanics work the alias is meant for. Going line-by-line:

- `succ = "uses"` This is the success case that will override this variable to indicate a successful use instead of a failed attempt.
- `D = desc` This just sets one variable to another. The alias will use `D` later when showing the result to the player
- `ch.mod_cc(cc, -1)` This will modify (`mod_cc`) the value of the counter (`cc`) by `-1`, reducing it from 1 to 0 and marking it as used
- `ch.set_hp(1)` This sets the character's hitpoints to 1.

```
elif ch.hp:
    D = hasHP
elif ch.cc_exists(cc):
    D = rest
else:
    D = noCC
```

And this introduces a little more complexity to if-blocks! The previous if-check defined the conditions for the ability succeeding. If one or more of those conditions failed, that block would be skipped and these conditions will be checked, in order, until one succeeds. If the initial `if` and all of the `elif` conditions fail, the `else` will run.

After this whole `if ... elif ... else` block is finished, `D` will contain the body text of the embed, and will be one of the 4 response strings that were defined above:

1) it works (desc)
2) you have more than 0 hp (hasHP)
3) you already used the feature (rest)
4) you don't have the counter in the first place (noCC)

```
T = f"{name} {succ} {cc}!"
F = f"{cc}|{ch.cc_str(cc) if ch.cc_exists(cc) else '*None*'}"
```

Setting some more variables that will be used in the embed. T will be used in the title of the embed, indicating either success or failure to the player. F will be the contents of a Field that will include the value of the counter in the embed (or `*None*` if the character doesn't have the counter). They use fstrings, or formatted strings, to streamline the code a bit.

```
</drac2>
```

This closes off the code block and everything else will be arguments to the embed command.

```
-title "{{T}}"
-desc "{{D}}"
-f "{{F}}"
```

This will send the defined variables to the embed to be displayed.

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

This makes it look pretty, setting the embed color and the character's image (if any) as a thumbnail

The end result is:

```
!alias orc-relentless embed
<drac2>
#Define variables for later use
cc = "Relentless Endurance"
desc = "When you are reduced to 0 hit points but not killed outright, you can drop to 1␣
↪hit point instead."
rest = "You can't use this feature again until you finish a long rest."
hasHP = "You have not been reduced to 0 hit points."
noCC = "You do not have this ability."
ch=character()

#Create the counter if it should exist but doesn't already
if ch.race.lower() == "half-orc":
    ch.create_cc_nx(cc, 0, 1, "long", "bubble", None, None, cc, desc+" "+rest)

#Logic of the alias. Check for all the necessary conditions
succ = "tries to use"
if ch.cc_exists(cc) and ch.get_cc(cc) and not ch.hp:
    succ = "uses"
    D = desc
    ch.mod_cc(cc, -1)
    ch.set_hp(1)
elif ch.hp:
    D = hasHP
elif ch.cc_exists(cc):
    D = rest
else:
    D = noCC

#Prepare the output
T = f"{name} {succ} {cc}!"
F = f"{cc}|{ch.cc_str(cc) if ch.cc_exists(cc) else '*None*'}"
</drac2>
-title "{{T}}"
-desc "{{D}}"
-f "{{F}}"
-color <color>
-thumb <image>
```

8.2 Insult Tutorial

By @Croebh#5603 with minor drac2 updates by @mobius#1442

```
!servalias insult embed
```

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

```
<drac2>
```

This specifies the start of a code block.

```
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"]`

```
I = " ".join(I)
```

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

```
</drac2>
```

This closes off the code block and everything else will be arguments to the embed command.

```
-title "You {{I}}!"
```

This adds a -title to the embed the alias starts with. The contents of this 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
<drac2>
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]]
I = " ".join(I)
</drac2>
-title "You {{I}}!"
-thumb <image> -color <color>
```

CHAPTER

NINE

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!

9.1 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.

9.2 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.

9.2.1 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
```

9.2.2 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
```

9.2.3 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, otherwise it is cast to `str`.

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!
```

9.2.4 Draconic Blocks

Syntax

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

Description: Runs the multi-line Draconic code between the delimiters. If a non-`None` value is returned (via the `return` keyword), is replaced by the returned value, cast to `str`.

Examples

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

(continues on next page)

(continued from previous page)

```
... </drac2>
[0, 2, 4]
```

```
>>> !test <drac2>
... out = []
... for stat in ['strength', 'dexterity', 'constitution']:
...   out.append(get(stat))
... </drac2>
... My STR, DEX, and CON scores are {{out}}!
My STR, DEX, and CON scores are [12, 18, 14]!
```

9.2.5 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)!

Non-Code, Space-Aware

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

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

Non-Code, Preserving All

Syntax: `%*%`

Description: Replaced with the unmodified string following the alias.

In Code, Quote-Escaping

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.

In Code, Quote-Escaping All

Syntax: `&*&`

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

In Code, List Literal

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']
```

9.3 Cvar Table

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

Name	Description	Type
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
dexterity	Dexterity score.	int
dexterityMod	Dexterity modifier.	int
dexteritySave	Dexterity saving throw modifier.	int
intelligence	Intelligence score.	int
intelligenceMod	Intelligence modifier.	int
intelligenceSave	Intelligence saving throw modifier.	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
armor	Armor Class.	int
color	The CSettings color for the character	str
description	Full character description.	str
hp	Maximum hit points.	int
image	Character image URL.	str
level	Character level.	int
name	The character's name.	str
proficiencyBonus	Proficiency bonus.	int
spell	The character's spellcasting ability mod.	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.

9.4 Function Reference

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

9.4.1 Python Builtins

abs(*x*)

Takes a number (float or int) and returns the absolute value of that number.

Parameters

x (*float or int*) – The number to find the absolute value of.

Returns
The absolute value of x.

Return type
float or int

`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[, start=0*)

Returns a iterable of tuples containing a count and the values from the iterable.

Parameters

- **x** (*iterable*) – The value to convert.
- **start** (*int*) – The starting value for the count

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

9.4.2 Draconic Functions

argparse(*args*, *parse_ephem=True*)

Given an argument string or list, returns the parsed arguments using the argument nondeterministic finite automaton.

If *parse_ephem* is False, arguments like `-d1` are saved literally rather than as an ephemeral argument.

Note: Arguments must begin with a letter and not end with a number (e.g. `d`, `e12s`, `a!!`). Values immediately following a flag argument (i.e. one that starts with `-`) will not be parsed as arguments unless they are also a flag argument.

There are three exceptions to this rule: `-i`, `-h`, and `-v`, none of which take additional values.

Parameters

- **args** – A list or string of arguments.
- **parse_ephem** (*bool*) – Whether to treat args like `-d1` as ephemeral arguments or literal ones.

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.

load_yaml(*self*, *yamlstr*)

Loads an object from a YAML string. See yaml.safe_load.

dump_yaml(*self*, *obj*, *indent=2*)

Serializes an object to a YAML string. See yaml.safe_dump.

load_json(*self*, *jsonstr*)

Loads an object from a JSON string. See `json.loads()`.

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*[, *default=None*])

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

`get_uvars`()

Retrieves and returns the dict of uvars.

Returns
A dict of all uvars.

Return type
dict

`get_uvar`(*name*[, *default=None*])

Retrieves and returns the value of a uvar (user variable).

Parameters

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

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

Return type
str or None

`randint`(*stop*)

`randint`(*start*, *stop*[, *step*])

Returns a random integer in the range `[start..stop)`.

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`.

Parameters

- **start** (*int*) – The lower limit (inclusive).
- **stop** (*int*) – The upper limit (non-inclusive).
- **step** (*int*) – The step value.

Returns
A random integer.

Return type
int

randchoice(*seq*)

Returns a random item from `seq`.

Parameters
seq (*iterable.*) – The itterable to choose a random item from.

Returns
A random item from the iterable.

Return type
Any.

randchoices(*population*, *weights=None*, *cum_weights=None*, *k=1*)

Returns a list of random items from `population` of `k` length with either weighted or cumulatively weighted odds. The `weights` [2,1,1] are equal to `cum_weights` [2,3,4]. If no `weights` or `cum_weights` are input, the items in `population` will have equal odds of being chosen. If no `k` is input, the output length will be 1.

Parameters

- **population** (*iterable.*) – The itterable to choose random items from.
- **weights** (*list of integers, floats, and fractions but not decimals*) – The odds for each item in the `population` iterable.
- **cum_weights** (*list of integers, floats, and fractions but not decimals*) – The cumulative odds for each item in the `population` itterable.
- **k** (*int*) – The length of the output.

Returns
A list of random items from the iterable.

Return type
list

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.

signature(*data=0*)

Returns a unique signature encoding the time the alias was invoked, the channel it was invoked in, the invoker, the id of the workshop collection the alias originates from (if applicable), and whether the caller was a personal/server alias/snippet.

This signature is signed with a secret that guarantees that valid signatures cannot be spoofed; use this when it is necessary to audit the invocations of an alias (e.g. to ensure that a server version of the alias is being used over a personal version).

Use *verify_signature()* in a separate alias to verify the integrity of a generated signature and unpack the encoded data.

Parameters

data (*int*) – Some user-supplied data to encode in the signature. This must be an unsigned integer that fits within 5 bits (i.e. a value [0..31]). Default 0.

Return type

str

verify_signature(*data*)

Verifies that a given signature is valid. The signature should have been generated by *signature()*.

If the signature is not valid, raises a ValueError. Otherwise, returns a dict with the following keys representing the context the given signature was generated in:

```
{
    "message_id": int,
    "channel_id": int,
    "author_id": int,
    "timestamp": float,
    "workshop_collection_id": str?,  # None if the caller was not a workshop object
    "scope": str,  # one of UNKNOWN, PERSONAL_ALIAS, SERVER_ALIAS, PERSONAL_SNIPPET,
↪ SERVER_SNIPPET, COMMAND_TEST
    "user_data": int,
    "guild_id": int?,  # may be None
    "guild": AliasGuild?,  # may be None
    "channel": AliasChannel?,  # may be None
    "author": AliasAuthor?,  # may be None
}
```

Parameters

data (*str*) – The signature to verify.

Return type

dict

If you are building your own application and want to verify these signatures yourself, we provide an API endpoint you can use to verify signatures!

Below is an example of Python code to verify a signature using the `httpx` (requests-like) library:

```
signature = "Dc3SEuDEMKIJZ0qbasAAAQKZ2xjlQgAAAAAAAAAAAAAAAAAAABQ==.
↪B5RLdufsD9utKaDou+94LEfOgpA="
async with httpx.AsyncClient() as client:
```

(continues on next page)

(continued from previous page)

```
    r = await client.post(
        "https://api.avrae.io/bot/signature/verify",
        json={"signature": signature}
    )
print(r.json(indent=2))
```

The endpoint's response model is the same as `verify_signature()` sans the `guild`, `channel`, and `author` keys (IDs are still present).

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

using(*self*, ***imports*)

Imports Draconic global variables as modules in the current namespace. See *Using Imports* for details.

Usually this should be the first statement in a code block if imports are used.

Warning: Only import modules from trusted sources! The entire contents of an imported module is executed once upon import, and can do bad things like delete all of your variables.

```
>>> using(
...     hello="50943a96-381b-427e-adb9-eea8ebf61f27"
... )
>>> hello.hello()
"Hello world!"
```

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

parse_coins(*args: str*) → dict

Parses a coin string into a representation of each currency. If the user input is a decimal number, assumes gold pieces. Otherwise, allows the user to specify currencies in the form '+1gp -2sp 3cp'

Parameters

- **args** (*str*) – The coin string to parse
- **include_total** (*bool*) – Whether to include the *"total"* key

Returns

A dict of the coin changes, e.g. `{"pp":0, "gp":1, "ep":0, "sp":-2, "cp":3, "total": 0.83}`

Return type

dict

9.5 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
Init Metadata	Yes	Yes	Explicit	Initiative	Anyone in channel

9.5.1 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 at the runtime's instantiation.

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.

9.5.2 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 at the runtime's instantiation. 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.

9.5.3 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".

9.5.4 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.

9.5.5 Honorable Mention: Initiative Metadata

Initiative metadata is a form of key-value pair storage attached to an ongoing initiative in a given channel. This storage is usually used for storing a medium-sized amount of programmatic information about an ongoing initiative (e.g. an alias' metadata on each combatant).

Metadata can be created, retrieved, and deleted using the *SimpleCombat.set_metadata()*, *SimpleCombat.get_metadata()*, and *SimpleCombat.delete_metadata()* methods, respectively.

9.6 Using Imports

Imports are a way for alias authors to share common code across multiple aliases, provide common libraries of code for other authors to write code compatible with your alias, and more!

If you already have the address of a module to import, use *using()* at the top of your code block in order to import the module into your namespace. For example:

```
!alias hello-world echo <drac2>
using(
    hello="50943a96-381b-427e-adb9-eea8ebf61f27"
)
return hello.hello()
</drac2>
```

Use `!gvar 50943a96-381b-427e-adb9-eea8ebf61f27` to take a peek at the `hello` module!

You can also import multiple modules in the same expression:

```
!alias hello-world echo <drac2>
using(
    hello="50943a96-381b-427e-adb9-eea8ebf61f27",
    hello_utils="0bbddb9f-c86f-4af8-9e04-1964425b1554"
)
return f"{hello.hello('you')}\n{hello_utils.hello_to_my_character()}"
</drac2>
```

The `hello_utils` module (`!gvar 0bbddb9f-c86f-4af8-9e04-1964425b1554`) also demonstrates how modules can import other modules!

Each imported module is bound to a namespace that contains each of the names (constants, functions, etc) defined in the module. For example, the `hello` module (`50943a96-381b-427e-adb9-eea8ebf61f27`) defines the `HELLO_WORLD` constant and `hello()` function, so a consumer could access these with `hello.HELLO_WORLD` and `hello.hello()`, respectively.

Warning: Only import modules from trusted sources! The entire contents of an imported module is executed once upon import, and can do bad things like delete all of your variables.

All gvar modules are open-source by default, so it is encouraged to view the imported module using `!gvar`.

Note: Modules do not have access to the argument parsing special syntax (i.e. `&ARGS&`, `%1%`, etc), and the variables listed in the Cvar Table are not implicitly bound in a module's execution.

9.6.1 Writing Modules

Modules are easy to publish and update! Simply create a gvar that contains valid Draconic code (**without** wrapping it in any delimiters such as <drac2>).

We encourage modules to follow the following format to make them easy to read:

```
# recommended_module_name
# This is a short description about what the module does.
#
# SOME_CONSTANT: some documentation about what this constant is
# some_function(show, the, args): some short documentation about what this function does
#     and how to call it
#     wow, this is long! use indentation if you need multiple lines
#     but otherwise longer documentation should go in the function's """docstring"""

SOME_CONSTANT = 3.141592

def some_function(show, the, args):
    """Here is where the longer documentation about the function can go."""
    pass
```

Use `!gvar 50943a96-381b-427e-adb9-eea8ebf61f27` and `!gvar 0bbddb9f-c86f-4af8-9e04-1964425b1554` to view the `hello` and `hello_utils` example modules used above for an example!

Note: Because all gvars are public to anyone who knows the address, modules are open-source by default.

9.7 Catching Exceptions

Draconic supports a modified version of Python's exception handling ("try-except") syntax, the most significant difference being that exceptions must be caught explicitly by passing the *exact name* of the exception type to the `except` clause as a string or tuple of strings. A bare `except` may also be used to catch any exception in the `try` block.

For example, to cast an arbitrary string to an integer and catch errors raised by `int()`:

```
!test <drac2>
some_string = "123"
try:
    return int(some_string)
except ("ValueError", "TypeError"):
    return "I couldn't parse an int!"
</drac2>
```

Note: Unlike Python, only the exact exception type given by a string will be matched, without subclass checking.

Draconic `try` statements also support `else` and `finally` blocks, similar to Python.

9.8 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
- Functions
- Lambda Expressions
- Argument Unpacking

9.9 Initiative Models

9.9.1 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.

name

The name of the combat (*str*), or `None` if no custom name is set.

round_num

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

turn_num

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

delete_metadata(*k: str*) → Optional[str]

Removes a key from the metadata.

Parameters
k (*str*) – The metadata key to remove

Returns
The removed value or `None` if the key is not found.

Return type
str or None

```
>>> delete_metadata("Test")
'{"Status": ["Mario", 1, 2]}'
```

end_round()

Moves initiative to just before the next round (no active combatant or group). Ending the round will not tick any events with durations.

get_combatant(*name*, *strict=None*)

Gets a combatant by its name or ID.

Parameters

- **name** (*str*) – The name or id of the combatant or group to get.
- **strict** – Whether combatant name must be a full case insensitive match. If this is `None` (default), attempts a strict match with fallback to partial match. If this is `False`, it returns the first partial match. If this is `True`, it will only return a strict match.

Returns
The combatant or group or None.

Return type
SimpleCombatant or *~aliasing.api.combat.SimpleGroup*

get_group(*name*, *strict=None*)

Gets a *SimpleGroup* that matches on name.

Parameters

- **name** (*str*) – The name of the group to get.
- **strict** – Whether combatant name must be a full case insensitive match. If this is `None` (default), attempts a strict match with fallback to partial match. If this is `False`, it returns the first partial match. If this is `True`, it will only return a strict match.

Returns
The group or None.

Return type
SimpleGroup

get_metadata(*k: str*, *default=None*) → str

Gets a metadata value for the passed key or returns *default* if the name is not set.

Parameters

- **k** (*str*) – The metadata key to get
- **default** – What to return if the name is not set.

```
>>> get_metadata("Test")
'{"Status": ["Mario", 1, 2]}'
```

set_metadata(*k: str*, *v: str*)

Assigns a metadata key to the passed value. Maximum size of the metadata is 100k characters, key and item inclusive.

Parameters

- **k** (*str*) – The metadata key to set
- **v** (*str*) – The metadata value to set

```
>>> set_metadata("Test", dump_json({"Status": ["Mario", 1, 2]}))
```

set_round(*round_num: int*)

Sets the current round. Setting the round will not tick any events with durations.

Parameters
round_num (*int*) – the new round number

9.9.2 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

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: Optional[str] = None*, *duration: Optional[int] = None*, *concentration: bool = False*, *parent: Optional[*SimpleEffect*] = None*, *end: bool = False*, *desc: Optional[str] = None*, *passive_effects: Optional[dict] = None*, *attacks: Optional[list[dict]] = None*, *buttons: Optional[list[dict]] = None*, *tick_on_combatant_id: Optional[str] = None*)

Adds an effect to the combatant. Returns the added effect.

Note: It is recommended to pass all arguments other than *name* to this method as keyword arguments (i.e. `add_effect("On Fire", duration=10)`). This is not strictly enforced for backwards-compatibility.

Warning: The *args* argument is deprecated as of v4.1.0. Use *passive_effects* instead.

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. Pass `None` for indefinite.
- **concentration** (*bool*) – Whether the effect requires concentration.
- **parent** (*SimpleEffect*) – The parent of the effect.

- **end** (*bool*) – Whether the effect ends on the end of turn.
- **desc** (*str*) – A description of the effect.
- **passive_effects** – The passive effects this effect should grant. See *Initiative Effect Args*.
- **attacks** – The attacks granted by this effect. See *Initiative Effect Args*.
- **buttons** – The buttons granted by this effect. See *Initiative Effect Args*.
- **tick_on_combatant_id** – The ID of the combatant whose turn the effect duration ticks on (defaults to the combatant who the effect is on).

Return type
SimpleEffect

property attacks

The attacks of the creature.

Return type
AliasAttackList

property controller

The ID of the combatant's controller.

Return type
int

property creature_type

The creature type of the creature. Will return None for players or creatures with no creature type.

Return type
str or None

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*, *strict: bool = False*)

Gets a SimpleEffect, fuzzy searching (partial match) for the first match or an exact match.

Parameters

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

- **strict** (*bool*) – Whether effect name must be an exact match. If this is `False`, it returns the first partial match. If this is `True`, it will only return a strict match.

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 id

The combatant's unique identifier.

Return type
str

property is_hidden

Whether the HP, AC, Resists, etc are hidden.

Return type
bool

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.

Returns
A string describing the creature's current, max, and temp HP after the change.

Return type
str

`property monster_name`

The monster name of the combatant. Will return None for players.

Return type
str or None

`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

`property race`

The race of the combatant. Will return None for monsters or combatants with no race.

Return type
str or None

`remove_effect`(*name: str*, *strict: bool = False*)

Removes an effect from the combatant, fuzzy searching on name or an exact match. If not found, does nothing.

Parameters

- **name** (*str*) – The name of the effect to remove.
- **strict** (*bool*) – Whether effect name must be an exact match. If this is `False`, it returns the first partial match. If this is `True`, it will only return a strict match.

`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: Optional[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).

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

9.9.3 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

init

What the group rolled for initiative.

Type
int

get_combatant(*name*, *strict=None*)

Gets a *SimpleCombatant* from the group.

param str name
The name of the combatant to get.

param strict
Whether combatant name must be a full case insensitive match. If this is `None` (default), attempts a strict match with fallback to partial match. If this is `False`, it returns the first partial match. If this is `True`, it will only return a strict match.

return
The combatant or None.

rtype
SimpleCombatant

property id

The group's unique identifier.

Return type
str

property name

The name of the group.

Return type
str

set_init(*init: int*)

Sets the group's initiative roll.

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

9.9.4 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. `None` if the effect has indefinite duration.

Type
int or None

effect

The applied passive effects of the object:

```
{
    attack_advantage: int
    to_hit_bonus: str
    damage_bonus: str
    magical_damage: bool
    silvered_damage: bool
```

(continues on next page)

(continued from previous page)

```
    resistances: List[Resistance]
    immunities: List[Resistance]
    vulnerabilities: List[Resistance]
    ignored_resistances: List[Resistance]
    ac_value: int
    ac_bonus: int
    max_hp_value: int
    max_hp_bonus: int
    save_bonus: str
    save_adv: List[str]
    save_dis: List[str]
    check_bonus: str
}
```

Each attribute in the dictionary is optional and may not be present.

Type
dict

name

The name of the effect.

Type
str

remaining

The remaining duration of the effect, in rounds. `None` if the effect has indefinite duration.

Type
int or None

ticks_on_end

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

Type
bool

attacks

A list of the attacks granted by the effect.

Type
list

buttons

A list of the buttons granted by the effect.

Type
list

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.

9.9.5 Initiative Effect Args

The *passive_effects*, *attacks*, and *buttons* arguments to `SimpleCombatant.add_effect()` should be a dict/list that follows the schema below, respectively.

Some examples are provided below.

```
class PassiveEffects:
    attack_advantage: Optional[enums.AdvantageType]
    to_hit_bonus: Optional[str255]
    damage_bonus: Optional[str255]
    magical_damage: Optional[bool]
    silvered_damage: Optional[bool]
    resistances: Optional[List[str255]]
    immunities: Optional[List[str255]]
    vulnerabilities: Optional[List[str255]]
    ignored_resistances: Optional[List[str255]]
    ac_value: Optional[int]
    ac_bonus: Optional[int]
    max_hp_value: Optional[int]
    max_hp_bonus: Optional[int]
    save_bonus: Optional[str255]
    save_adv: Optional[Set[str]]
    save_dis: Optional[Set[str]]
    check_bonus: Optional[str255]
    check_adv: Optional[Set[str]]
    check_dis: Optional[Set[str]]

class AttackInteraction:
    attack: AttackModel  # this can be any attack built on the Avrae Dashboard
    override_default_dc: Optional[int]
    override_default_attack_bonus: Optional[int]
    override_default_casting_mod: Optional[int]

class ButtonInteraction:
    automation: Automation  # this can be any automation built on the Avrae Dashboard
    label: str
    verb: Optional[str255]
    style: Optional[conint(ge=1, le=4)]
    override_default_dc: Optional[int]
    override_default_attack_bonus: Optional[int]
    override_default_casting_mod: Optional[int]
```

Example: Passive Effects

Also see *PassiveEffects* for more information.

```
combatant.add_effect(
    "Some Magical Effect",
    passive_effects={
        "attack_advantage": 1,
        "damage_bonus": "1d4 [fire]",
        "magical_damage": True,
        "resistances": ["fire", "nonmagical slashing"],
        "ac_bonus": 2,
        "save_adv": ["dexterity"]
    }
)
```

Example: Granting Attacks

Also see *AttackInteraction* for more information. Note that the Automation schema differs slightly from the aliasing API.

```
combatant.add_effect(
    "Some Magical Effect",
    attacks=[{
        "attack": {
            "_v": 2,
            "name": "Magical Attack",
            "verb": "shows off the power of",
            "automation": [
                {
                    "type": "target",
                    "target": "each",
                    "effects": [
                        {
                            "type": "attack",
                            "hit": [
                                {
                                    "type": "damage",
                                    "damage": "1d10[fire]"
                                }
                            ],
                            "miss": []
                        }
                    ]
                }
            ]
        }
    }]
)
```

Example: Granting Buttons

Also see *ButtonInteraction* for more information. Note that the Automation schema differs slightly from the aliasing API.

```
combatant.add_effect(
    "Some Magical Effect",
    buttons=[{
```

(continues on next page)

(continued from previous page)

```
        "label": "On Fire",
        "verb": "is burning",
        "style": 4,
        "automation": [
            {
                "type": "target",
                "target": "self",
                "effects": [
                    {
                        "type": "damage",
                        "damage": "1d6 [fire]"
                    }
                ]
            }
        ]
    }]
)
```

9.10 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`.

9.11 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*, or dict[str, list[str]]) – The arguments to add.

adv(*eadv=False*, *boolwise=False*, *ephem=False*, *custom: Optional[dict] = None*)

Determines whether to roll with advantage, disadvantage, Elven Accuracy, or no special effect.

Parameters

- **eadv** – 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 eadv

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_** (*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_=<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_** (*type*) – The type that the arg should be returned as.
- **ephem** (*bool*) – 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.

9.12 Context Models

9.12.1 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 message_id

The ID of the message the alias was invoked with.

```
>>> !test {{ctx.message_id}}
982495360129847306
```

Return type

int

property prefix

The prefix used to run the alias.

Return type
str

9.12.2 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

9.12.3 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 parent

If this channel is a thread, the thread's parent channel, or None otherwise.

Return type
AliasChannel or None

property topic

The channel topic. This will be None if the channel is a direct message or thread.

Return type
str or None

9.12.4 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

9.12.5 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

get_roles()

The user's roles. When used in a DM, it is always empty.

Return type
list of `AliasRole`

property id

The user's ID.

Return type
int

property name

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

Return type
str

9.13 AliasCharacter

class AliasCharacter(*AliasStatBlock*)

property ac

The armor class of the creature.

Return type
int or None

property actions

The character's actions. These do not include attacks - see the `attacks` property.

Return type
list[*AliasAction*]

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 coinpurse

The coinpurse of the character.

Return type
AliasCoinpurse

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`, `'bubble'`, `'square'`, `'hex'`, or `'star'`.
- **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 annotated dice strings.
- **title** (*str*) – The title of the counter.
- **desc** (*str*) – The description of the counter.
- **initial_value** (*str*) – The initial value of the counter.

Return type
AliasCustomCounter

Returns
The newly created counter.

create_cc_nx(*name: str*, *minVal: Optional[str] = None*, *maxVal: Optional[str] = None*, *reset: Optional[str] = None*, *dispType: Optional[str] = None*, *reset_to: Optional[str] = None*, *reset_by: Optional[str] = None*, *title: Optional[str] = None*, *desc: Optional[str] = None*, *initial_value: Optional[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 creature_type

The creature type of the creature. Will return None for players or creatures with no creature type.

Return type
str or None

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 character's 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.

Note: This method does not unbind the name in the current runtime.

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

property description

The description of the character.

Return type
str or None

edit_cc(*name: str*, *minVal: str = <object object>*, *maxVal: str = <object object>*, *reset: str = <object object>*, *dispType: str = <object object>*, *reset_to: str = <object object>*, *reset_by: str = <object object>*, *title: str = <object object>*, *desc: str = <object object>*, *new_name: ~typing.Optional[str] = None*)

Edits an existing custom counter.

Pass `None` to remove an argument entirely. Will clamp counter value to new limits if needed.

Parameters

- **name** (*str*) – The name of the counter to edit.
- **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`, `'bubble'`, `'square'`, `'hex'`, or `'star'`.
- **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 annotated dice strings.
- **title** (*str*) – The title of the counter.
- **desc** (*str*) – The description of the counter.
- **new_name** (*str*) – The new name of the counter.

Return type
AliasCustomCounter

Raises
`ConsumableException` if the counter does not exist.

Returns
The edited counter

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.

`get_cvar`(*name*, *default=None*)
Retrieves and returns the value of a cvar (character variable).

Parameters

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

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

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 image`
The image url for the character.

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.

Returns

A string describing the creature's current, max, and temp HP after the change.

Return type

str

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. Binds the value to the given name in the current runtime.

Parameters

- **name** (*str*) – The name of the variable to set. Must be a valid identifier and not be in the *Cvar Table*.
- **val** (*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*.
- **val** (*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, dicecloudv2, 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

9.13.1 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', 'square', 'hex', or 'star')

Return type
str

full_str(*include_name: bool = False*)

Returns a string representing the full custom counter.

Parameters
include_name (*bool*) – If the name of the counter should be included. Defaults to False.

Returns
A string representing all components of the counter.

Return type
str

Example:

```
>>> full_str()
"\n"
"**Resets On**: Long Rest"
>>> full_str(True)
"**Bardic Inspiration**\n"
"\n"
"**Resets On**: Long Rest"
```

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

mod(*value*, *strict=False*)

Modifies the value of the custom counter.

Parameters

- **value** (*int*) – The value to modify the custom counter by.
- **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 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

9.13.2 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

9.13.3 AliasAction

`class AliasAction`

An action.

`property activation_type`

The activation type of the action (e.g. action, bonus, etc).

Action Type	Value
Action	1
No Action	2
Bonus Action	3
Reaction	4
Minute	6
Hour	7
Special	8
Legendary Action	9
Mythic Action	10
Lair Action	11

Return type
int or None

`property activation_type_name`

The name of the activation type of the action. Will be one of: "ACTION", "NO_ACTION", "BONUS_ACTION", "REACTION", "MINUTE", "HOUR", "SPECIAL". This list of options may expand in the future.

Return type
str

`property description`

The description of the action as it appears in a non-verbose action list.

Return type
str

`property name`

The name of the action.

Return type
str

property snippet

The description of the action as it appears in a verbose action list.

Return type
str

9.13.4 AliasCoinpurse

class AliasCoinpurse

An object holding the coinpurse for the active character.

str(*AliasCoinpurse*)

Returns a string representation of the entire coinpurse. If the character setting for Compact Coins is enabled, this will only return your float gold, otherwise will return all 5 coin types.

Type
str

pp

gp

ep

sp

cp

The value of the given coin type.

Type
int

AliasCoinpurse[cointype]

Gets the value of the given coin type.

Type
int

autoconvert()

Converts all of your coins into the highest value coins possible. 100cp turns into 1gp, 5sp turns into 1ep, etc.

coin_str(*cointype: str*) → str

Returns a string representation of the chosen coin type.

Parameters
cointype (*str*) – The type of coin to return. "pp", "gp", "ep", "sp", and "cp"

Returns
The string representation of the chosen coin type.

Return type
str

compact_str() → str

Returns a string representation of the compacted coin value.

Returns
The string representation of the compacted coin value.

Return type
str

`get_coins`() → dict

Returns a dict of your current coinpurse.

Returns
A dict of your current coinpurse, e.g. `{"pp":0, "gp":1, "ep":0, "sp":2, "cp":3, "total": 1.23}`

Return type
dict

`modify_coins`(*pp: int = 0*, *gp: int = 0*, *ep: int = 0*, *sp: int = 0*, *cp: int = 0*, *autoconvert: bool = True*)

Modifies your coinpurse based on the provided values. If `autoconvert` is enabled, it will convert down higher value coins if necessary to handle the transaction. Returns a dict representation of the deltas.

Parameters

- **pp** (*int*) – Platinum Pieces. Defaults to `0`.
- **gp** (*int*) – Gold Pieces. Defaults to `0`.
- **ep** (*int*) – Electrum Pieces. Defaults to `0`.
- **sp** (*int*) – Silver Pieces. Defaults to `0`.
- **cp** (*int*) – Copper Pieces. Defaults to `0`.
- **autoconvert** (*bool*) – Whether it should attempt to convert down higher value coins. Defaults to `True`

Returns
A dict representation of the delta changes for each coin type.

Return type
dict

`set_coins`(*pp: int*, *gp: int*, *ep: int*, *sp: int*, *cp: int*)

Sets your coinpurse to the provided values.

Parameters

- **pp** (*int*) – Platinum Pieces
- **gp** (*int*) – Gold Pieces
- **ep** (*int*) – Electrum Pieces
- **sp** (*int*) – Silver Pieces
- **cp** (*int*) – Copper Pieces

`property total: float`

Returns the total amount of coins in your bag, converted to float gold.

Return type
float

9.14 StatBlock Models

9.14.1 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 creature_type

The creature type of the creature. Will return None for players or creatures with no creature type.

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

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.

Returns
A string describing the creature's current, max, and temp HP after the change.

Return type
str

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

9.14.2 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(*stat*)

Get the integer value of a stat (case sensitive, lowercase. strength, dexterity, etc).

Parameters
stat (*str*) – The stat to look up

Returns
The integer value of the stat.

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

9.14.3 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

9.14.4 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

9.14.5 AliasAttack

`class AliasAttack`

An attack.

`str`(*AliasAttack*)

Returns a string representation of this attack.

Type
str

`property activation_type`

The activation type of the action (e.g. action, bonus, etc).

Action Type	Value
Action	1
No Action	2
Bonus Action	3
Reaction	4
Minute	6
Hour	7
Special	8
Legendary Action	9
Mythic Action	10
Lair Action	11

Return type
int

`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

9.14.6 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

9.14.7 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

9.14.8 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

9.14.9 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*]

is_immune(*damage_type: str*) → bool

Whether or not this AliasResistances contains any immunities that apply to the given damage type string.

If the AliasResistances contains both a neutral and an immunity that applies, returns False.

is_neutral(*damage_type: str*) → bool

Whether or not this AliasResistances contains any neutrals that apply to the given damage type string.

is_resistant(*damage_type: str*) → bool

Whether or not this AliasResistances contains any resistances that apply to the given damage type string.

If the AliasResistances contains both a neutral and a resistance that applies, returns False.

is_vulnerable(*damage_type: str*) → bool

Whether or not this AliasResistances contains any vulnerabilities that apply to the given damage type string.

If the AliasResistances contains both a neutral and a vulnerability that applies, returns False.

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*]

9.14.10 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

property is_complex

Whether or not the resistance has some more complex conditional beyond just a dtype.

9.14.11 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

find(*spell_name: str*)

Returns a list of the spells of the given name in the spellbook, case-insensitive.

Return type
List[*AliasSpellbookSpell*]

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, including pact 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, including pact slots.

`property max_pact_slots`

The maximum number of pact slots the spellcaster has remaining. If the spellcaster has no pact slots, returns None.

Note: Only D&D Beyond character sheets support the explicit distinction between pact and non-pact slots.

Return type
int or None

`property num_pact_slots`

The number of pact slots the spellcaster has remaining. If the spellcaster has no pact slots, returns None.

Note: Only D&D Beyond character sheets support the explicit distinction between pact and non-pact slots.

Return type
int or None

`property pact_slot_level`

The spellcaster's pact slot level. If the spellcaster has no pact slots, returns None.

Note: Only D&D Beyond character sheets support the explicit distinction between pact and non-pact slots.

Return type
int or None

`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_pact_slots`()

Resets the number of remaining pact slots to the max, leaving non-pact slots untouched.

`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*, *pact=True*)

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.
- **pact** (*bool*) – Whether to prefer modifying Pact Magic slots (if applicable) or normal spell 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, including pact 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 prepared

Whether or not the spell is prepared. If the spell is always prepared, the caster is not a prepared caster (e.g. Sorcerer), or the spell is a cantrip, this will be True.

Return type
bool

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

TEN

AUTOMATION REFERENCE

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

10.1 Basic Structure

An automation run is made up of a list of *effects* (AKA *automation node*), each of which may have additional *effects* that it runs under certain conditions. This recursive structure is called the *automation tree*.

Glossary

`automation engine`

The Automation Engine is the code responsible for reading the automation tree and executing it against the current game state. It handles rolling the dice, checking against your targets' armor classes, modifying hit points, and other game mechanics.

`automation tree`

`automation`

The automation tree (sometimes just called "automation") is the program that the Automation Engine runs. It's made up of multiple nodes that all link together to make an attack, action, spell, or more.

`effect`

`node`

A single step of automation - usually, this is a D&D game mechanic like *rolling to hit*, *making a saving throw*, or *dealing damage*, but this can also be used in more programmatic ways to help set up other nodes.

10.2 Runtime Variables

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).
- `spell_attack_bonus` (*`int`* or None) - The attack bonus for the spell, or the caster's default attack bonus.
- `spell_dc` (*`int`* or None) - The DC for the spell, or the caster's default DC.
- `spell_level` (*`int`* or None) - The level used to cast the spell, or None
- `choice` (*`str`*) - The input provided by the `-choice` argument, always lowercase. If the arg was not used, it will be an empty string.

Additionally, runs triggered by an initiative effect (such as automation provided in a *ButtonInteraction*) provide the following variables:

- `ieffect` (*SimpleEffect*) The initiative effect responsible for providing the automation.

10.3 Target

```
{
    type: "target";
    target: "all" | "each" | int | "self" | "parent" | "children";
    effects: Effect[];
    sortBy?: "hp_asc" | "hp_desc";
    self_target?: boolean;
}
```

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

class Target

target

- `"all"` or `"each"` (actions only): Affects each of the given (by the `-t` argument) targets.
- `int` (actions only): Affects the Nth target (1-indexed).
- `"self"`: Affects the caster, or the actor the triggering effect is on if run from an IEffect button.
- `"parent"` (IEffect buttons only): If the triggering effect has a parent effect, affects the actor the parent effect is on.
- `"children"` (IEffect buttons only): If the triggering effect has any children effects, affects each actor a child effect is on.

effects

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

sortBy

optional - Whether to sort the target list. If not given, targets are processed in the order the `-t` arguments are seen. This does not affect `self` targets.

- `hp_asc`: Sorts the targets in order of remaining hit points ascending (lowest HP first, None last).
- `hp_desc`: Sorts the targets in order of remaining hit points descending (highest HP first, None last).

Variables

- `target` (*AliasStatBlock*) The current target.
- `targetIteration` (*int*) If running multiple iterations (i.e. `-rr`), the current iteration (1-indexed).
- `targetIterations` (*int*) The total number of iterations. Minimum 1, maximum 25.
- `targetIndex` (*int*) The index of the target in the list of targets processed by this effect (0-indexed - first target = `0`, second = `1`, etc.). Self targets, nth-targets, and parent targets will always be `0`.
- `targetNumber` (*int*) Same as `targetIndex`, but 1-indexed (equivalent to `targetIndex + 1`).

10.4 Attack

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

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

Attack:

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).

adv

optional - An IntExpression that details whether the attack has inherent advantage or not. `0` for flat, `1` for Advantage, `2` for Elven Accuracy, `-1` for Disadvantage (Default is flat).

Variables

- `lastAttackDidHit` (`bool`) Whether the attack hit.
- `lastAttackDidCrit` (`bool`) If the attack hit, whether it crit.
- `lastAttackRollTotal` (*`int`*) The result of the last to-hit roll (0 if no roll was made).
- `lastAttackNaturalRoll` (*`int`*) The natural roll of the last to-hit roll (e.g. *10* in *1d20 (10) + 5 = 15*; 0 if no roll was made).
- `lastAttackHadAdvantage` (*`int`*) The advantage type of the last to-hit roll. `0` for flat, `1` for; Advantage, `2` for Elven Accuracy, `-1` for Disadvantage

10.5 Save

```
{
    type: "save";
    stat: "str" | "dex" | "con" | "int" | "wis" | "cha";
    fail: Effect[];
    success: Effect[];
    dc?: IntExpression;
    adv?: -1 | 0 | 1;
}
```

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

class Save

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).

adv

optional, default 0 - Whether the saving throw should have advantage by default (`-1` = disadvantage, `1` = advantage, `0` = no advantage).

Variables

- `lastSaveDidPass` (`bool`) Whether the target passed the save.
- `lastSaveDC` (*int*) The DC of the last save roll.
- `lastSaveRollTotal` (*int*) The result of the last save roll (0 if no roll was made).
- `lastSaveNaturalRoll` (*int*) The natural roll of the last save roll (e.g. `10` in `1d20 (10) + 5 = 15`; 0 if no roll was made).
- `lastSaveAbility` (*str*) The title-case full name of the ability the save was made with (e.g. `"Strength"`, `"Wisdom"`, etc).

10.6 Damage

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

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

Note: This node can also be used to heal a target; simply use negative damage to supply healing.

class Damage

damage

How much damage to deal.

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.

fixedValue

optional - If `true`, won't add any bonuses to damage from `-d` arguments or damage bonus effects.

Variables

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

10.7 TempHP

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

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

class TempHP

amount

How much temp HP the target should have.

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.

10.8 IEffect

```
{
    type: "ieffect2";
    name: AnnotatedString;
    duration?: int | IntExpression;
    effects?: PassiveEffects;
    attacks?: AttackInteraction[];
    buttons?: ButtonInteraction[];
    end?: boolean;
    conc?: boolean;
    desc?: AnnotatedString;
    stacking?: boolean;
    save_as?: string;
```

(continues on next page)

(continued from previous page)

```
    parent?: string;
    target_self?: boolean;
    tick_on_caster?: boolean;
}
```

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

Note: If the targeted creature is not in combat, this will display the effects of the initiative effect but not save it on the creature.

---

class IEffect

name

The name of the effect to add. Annotations will show as *Variable* in the attack string.

duration

optional, default infinite - The duration of the effect, in rounds of combat. If this is negative, creates an effect with infinite duration.

Note: Wait, how do durations actually work?

Durations use a "tick" system, and `duration` is actually a measure of how many "ticks" an effect sticks around for. By default, each effect "ticks" once at the beginning of its combatant's turn.

By using `end` and `tick_on_caster`, you can control how the duration ticks in order to create effects that last until the end of your next turn, end of the caster's next turn, etc.

---

effects

optional, default no effects - The effects to add. See *PassiveEffects*.

attacks

optional, default no attacks - The attacks granted by this effect. See *AttackInteraction*.

buttons

optional, default no buttons - The buttons granted by this effect. See *ButtonInteraction*.

end

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

conc

optional, default false - Whether the effect requires concentration.

desc

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

stacking

optional, default false - If true and another effect with the same name is found on the target, instead of overwriting, add a child effect with name `{name} x{count}` and no description, duration, concentration, attacks, or buttons.

save_as

optional, default None - If supplied, saves an `IEffectMetaVar` to the automation runtime, which can be used in another IEffect's `parent` key to set its parent to this effect. Must be a valid identifier.

parent

optional, default None - If supplied, sets the created effect's parent to the given effect. This must be the name of an existing `IEffectMetaVar`.

If `parent` is supplied but the parent effect does not exist, will not set a parent.

If `conc` is true, the given parent effect will take priority over the concentration effect.

If `stacking` is true and a valid stack parent exists, the stack parent will take priority over the given parent.

target_self

optional, default false - If true, the effect will be applied to the caster of the action, rather than the target.

tick_on_caster

optional, default false - If true, the effect's duration will be dependent on the caster of the action, rather than the target. For example, a `tick_on_caster` effect with a duration of 1 will last until the start of the *caster's* next turn, rather than the *target's*.

If the caster is not in combat, this has no effect.

Variables

- `(supplied save_as)` (`IEffectMetaVar` or `None`) A reference to the effect that was added to the target. Use this in another IEffect's `parent` key to set that IEffect's parent to the given one.

10.8.1 PassiveEffects

```
{
    attack_advantage: IntExpression;
    to_hit_bonus: AnnotatedString;
    damage_bonus: AnnotatedString;
    magical_damage: IntExpression;
    silvered_damage: IntExpression;
    resistances: AnnotatedString[];
    immunities: AnnotatedString[];
    vulnerabilities: AnnotatedString[];
    ignored_resistances: AnnotatedString[];
    ac_value: IntExpression;
    ac_bonus: IntExpression;
    max_hp_value: IntExpression;
    max_hp_bonus: IntExpression;
    save_bonus: AnnotatedString;
    save_adv: AnnotatedString[];
    save_dis: AnnotatedString[];
    check_bonus: AnnotatedString;
    check_adv: AnnotatedString[];
    check_dis: AnnotatedString[];
    dc_bonus: IntExpression;
}
```

Used to specify the passive effects granted by an initiative effect.

class PassiveEffects

attack_advantage

optional, default no advantage - Whether this effect gives the combatant advantage on all attacks. -1 for dis, 1 for adv, 2 for elven accuracy.

`to_hit_bonus`

optional - A bonus that this effect grants to all of the combatant's to-hit rolls.

`damage_bonus`

optional - A bonus that this effect grants to all of the combatant's damage rolls.

`magical_damage`

optional, default false - Whether this effect makes all of the combatant's attacks do magical damage. 0 for false, anything else for true.

`silvered_damage`

optional, default false - Whether this effect makes all of the combatant's attacks do silvered damage. 0 for false, anything else for true.

`resistances`

optional - A list of damage types and optionally modifiers (e.g. "fire", "nonmagical slashing") that the combatant should be resistant to while this effect is active.

`immunities`

optional - A list of damage types and optionally modifiers (e.g. "fire", "nonmagical slashing") that the combatant should be immune to while this effect is active.

`vulnerabilities`

optional - A list of damage types and optionally modifiers (e.g. "fire", "nonmagical slashing") that the combatant should be vulnerable to while this effect is active.

`ignored_resistances`

optional - A list of damage types and optionally modifiers (e.g. "fire", "nonmagical slashing") that the combatant should *not* be resistant, immune, or vulnerable to while this effect is active.

`ac_value`

optional - A value to set the combatant's armor class to while this effect is active.

Note: If both `ac_value` and `ac_bonus` are specified, the resulting value will be equal to `ac_value + ac_bonus`.

If multiple effects specify `ac_value`, the highest value will be used.

`ac_bonus`

optional - A bonus added to the combatant's armor class while this effect is active.

`max_hp_value`

optional - A value to set the combatant's maximum hit points to while this effect is active.

Note: If both `max_hp_value` and `max_hp_bonus` are specified, the resulting value will be equal to `max_hp_value + max_hp_bonus`.

If multiple effects specify `max_hp_value`, the highest value will be used.

`max_hp_bonus`

optional - A bonus added to the combatant's maximum hit points while this effect is active.

`save_bonus`

optional - A bonus that this effect grants to all of the combatant's saving throws.

save_adv

optional - A list of stat names (e.g. `strength`) that the combatant should have advantage on for their respective saving throws while this effect is active. Use `all` as a stat name to specify all stats.

save_dis

optional - A list of stat names (e.g. `strength`) that the combatant should have disadvantage on for their respective saving throws while this effect is active. Use `all` as a stat name to specify all stats.

check_bonus

optional - A bonus that this effect grants to all of the combatant's skill checks.

check_adv

optional - A list of skill names (e.g. `sleightOfHand`, `strength`) that the combatant should have advantage on for ability checks for while this effect is active. If a base ability is given, the advantage will apply to all skills based on that ability (e.g. `strength` gives advantage on `athletics` checks). Use `all` as a stat name to specify all skills.

check_dis

optional - A list of skill names (e.g. `sleightOfHand`, `strength`) that the combatant should have disadvantage on for ability checks for while this effect is active. If a base ability is given, the disadvantage will apply to all skills based on that ability (e.g. `strength` gives disadvantage on `athletics` checks). Use `all` as a stat name to specify all skills.

dc_bonus

optional - A bonus added to the all of the combatant's save DCs while this effect is active.

10.8.2 AttackInteraction

```
{
    attack: Attack;
    defaultDC?: IntExpression;
    defaultAttackBonus?: IntExpression;
    defaultCastingMod?: IntExpression;
}
```

Used to specify an attack granted by an initiative effect: some automation that appears in the combatant's `!action list` and can be run with a command.

class AttackInteraction

attack

The Attack model is any valid individual entity as exported by the attack editor on the Avrae Dashboard. See *Custom Attack Structure*.

defaultDC

optional - The default saving throw DC to use when running the automation. If not provided, defaults to the targeted combatant's default spellcasting DC (or any DC specified in the automation). Use this if the effect's DC depends on the original caster's DC, rather than the target's DC.

defaultAttackBonus

optional - The default attack bonus to use when running the automation. If not provided, defaults to the targeted combatant's default attack bonus (or any attack bonus specified in the automation). Use this if the effect's attack bonus depends on the original caster's attack bonus, rather than the target's attack bonus.

defaultCastingMod

optional - The default spellcasting modifier to use when running the automation. If not provided, defaults to the targeted combatant's default spellcasting modifier. Use this if the effect's spellcasting modifier depends on the original caster's spellcasting modifier, rather than the target's spellcasting modifier.

10.8.3 ButtonInteraction

```
{
    automation: Effect[];
    label: AnnotatedString;
    verb?: AnnotatedString;
    style?: IntExpression;
    defaultDC?: IntExpression;
    defaultAttackBonus?: IntExpression;
    defaultCastingMod?: IntExpression;
}
```

Used to specify a button that will appear on the targeted combatant's turn and execute some automation when pressed.

Note: Any initiative effects applying an offensive effect to the caster will not be considered when a ButtonInteraction is run, to prevent scenarios where an effect granting a damage bonus to the caster increases the damage done by a damage over time effect and other similar scenarios.

You may think of this as a ButtonInteraction's caster being a temporary actor without any active initiative effects.

---

class ButtonInteraction

automation

The automation to run when this button is pressed.

label

The label displayed on the button.

verb

optional, default "uses {label}" - The verb to use for the displayed output when the button is pressed (e.g. "is on fire" would display "NAME is on fire!").

style

optional, default blurple - The color of the button (1 = blurple, 2 = grey, 3 = green, 4 = red).

defaultDC

optional - The default saving throw DC to use when running the automation. If not provided, defaults to the targeted combatant's default spellcasting DC (or any DC specified in the automation). Use this if the effect's DC depends on the original caster's DC, rather than the target's DC.

defaultAttackBonus

optional - The default attack bonus to use when running the automation. If not provided, defaults to the targeted combatant's default attack bonus (or any attack bonus specified in the automation). Use this if the effect's attack bonus depends on the original caster's attack bonus, rather than the target's attack bonus.

defaultCastingMod

optional - The default spellcasting modifier to use when running the automation. If not provided, defaults to the targeted combatant's default spellcasting modifier. Use this if the effect's spellcasting modifier depends on the original caster's spellcasting modifier, rather than the target's spellcasting modifier.

10.9 Remove IEffect

New in version 4.0.0.

```
{
    type: "remove_ieffect";
    removeParent?: "always" | "if_no_children";
}
```

Removes the initiative effect that triggered this automation. Only works when run in execution triggered by an initiative effect, such as a ButtonInteraction (see *ButtonInteraction*).

class RemoveIEffect

removeParent

optional, default null - If the removed effect has a parent, whether to remove the parent.

- `null` (default) - Do not remove the parent effect.
- `"always"` - If the removed effect has a parent, remove it too.
- `"if_no_children"` - If the removed effect has a parent and its only remaining child was the removed effect, remove it too.

Variables

No variables are exposed.

10.10 Roll

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

class Roll

dice

An AnnotatedString detailing what dice to roll.

name

The variable name 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 the `-d` argument.

displayName

The name to display in the Meta field. If left blank, it will use the saved name.

fixedValue

optional - If `true`, won't add any bonuses to damage from `-d` arguments or damage bonus effects.

Variables

- **(supplied name) (RollEffectMetaVar) The result of the roll.**
  - You can use this in an AnnotatedString to retrieve the simplified result of the roll. Using this variable in an AnnotatedString will always return a string that itself can be rolled.
  - You can use this in an IntExpression to retrieve the roll total.
  - You can compare this variable against a number to determine if the total of the roll equals that number.
- `lastRoll` (*int*) The integer total of the roll.

10.11 Text

```
{
    type: "text";
    text: AnnotatedString | AbilityReference;
    title: string
}
```

Outputs a short amount of text in the resulting embed.

class Text

text

Either:

- An AnnotatedString (the text to display).
- An AbilityReference (see *AbilityReference*). Displays the ability's description in whole.

title

optional - Allows you to set the name of the field. Defaults to "Effect"

10.12 Set Variable

New in version 2.7.0.

```
{
    type: "variable";
    name: string;
    value: IntExpression;
```

(continues on next page)

(continued from previous page)

```
    higher?: {int: IntExpression};
    onError?: IntExpression;
}
```

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

class SetVariable

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.

10.13 Condition (Branch)

New in version 2.7.0.

```
{
    type: "condition";
    condition: IntExpression;
    onTrue: Effect[];
    onFalse: Effect[];
    errorBehaviour?: "true" | "false" | "both" | "neither" | "raise";
}
```

Run certain effects if a certain condition is met, or other effects otherwise. AKA "branch" or "if-else".

class Condition

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
optional - 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.

10.14 Use Counter

New in version 2.10.0.

```
{
    type: "counter";
    counter: string | SpellSlotReference | AbilityReference;
    amount: IntExpression;
    allowOverflow?: boolean;
    errorBehaviour?: "warn" | "raise" | "ignore";
    fixedValue?: boolean;
}
```

Uses a number of charges of the given counter, and displays the remaining amount and delta.

Note: Regardless of the current target, this effect will always use the *caster's* counter/spell slots!

class UseCounter

counter

The name of the counter to use (case-sensitive, full match only), or a reference to a spell slot (see *SpellSlotReference*).

amount

The number of charges to use. If negative, will add charges instead of using them.

allowOverflow

optional, default False - If False, attempting to overflow/underflow a counter (i.e. use more charges than available or add charges exceeding max) will error instead of clipping to bounds.

errorBehaviour

optional, default "warn" - How to behave if modifying the counter raises an error:

- `"warn"`: Automation will continue to run, and any errors will appear in the output. (*default*)
- `"raise"`: Raise the error and halt execution.
- `"ignore"`: All errors are silently consumed.

Some, but not all, possible error conditions are:

- The target does not have counters (e.g. they are a monster)
- The counter does not exist
- `allowOverflow` is false and the new value is out of bounds

fixedValue

optional - If `true`, won't take into account `-amt` arguments.

Variables

- `lastCounterName` (*str*) The name of the last used counter. If it was a spell slot, the level of the slot (safe to cast to int, i.e. `int(lastCounterName)`). (`None` on error).
- `lastCounterRemaining` (*int*) The remaining charges of the last used counter (0 on error).
- `lastCounterUsedAmount` (*int*) The amount of the counter successfully used.

- `lastCounterRequestedAmount` (*int*) The amount of the counter requested to be used (i.e. the amount specified by automation or requested by `-amt`, regardless of the presence of the `-i` arg).

10.14.1 SpellSlotReference

```
{
    slot: number | IntExpression;
}
```

class SpellSlotReference

slot

The level of the spell slot to reference (`[1..9]`).

10.14.2 AbilityReference

```
{
    id: number;
    typeId: number;
}
```

In most cases, an `AbilityReference` should not be constructed manually; use the Automation editor to select an ability instead. A list of valid abilities can be retrieved from the API at `/gamedata/limiteduse`.

Note: The Automation Engine will make a best effort at discovering the appropriate counter to use for the given ability - in most cases this won't affect the chosen counter, but in some cases, it may lead to some unexpected behaviour. Some examples of counter discovery include:

- Choosing `Channel Divinity (Paladin)` may discover a counter granted by the Cleric's Channel Divinity feature
- Choosing `Breath Weapon (Gold)` may discover a counter for a breath weapon of a different color
- Choosing `Sorcery Points (Sorcerer)` may discover a counter granted by the Metamagic Adept feat

---

class AbilityReference

id

The ID of the ability referenced.

typeId

The DDB entity type ID of the ability referenced.

10.15 Cast Spell

New in version 2.11.0.

```
{
    type: "spell";
    id: int;
    level?: int;
    dc?: IntExpression;
    attackBonus?: IntExpression;
    castingMod?: IntExpression;
    parent?: string;
}
```

Executes the given spell's automation as if it were immediately cast. Does not use a spell slot to cast the spell. Can only be used at the root of automation. Cannot be used inside a spell's automation.

This is usually used in features that cast spells using alternate resources (i.e. Use Counter, Cast Spell).

class CastSpell

id

The DDB entity id of the spell to cast. Use the Automation Editor to select a spell or the `/gamedata/spells` API endpoint to retrieve a list of valid spell IDs.

level

optional - The (slot) level to cast the spell at.

dc

optional - The saving throw DC to use when casting the spell. If not provided, defaults to the caster's default spellcasting DC (or any DC specified in the spell automation).

attackBonus

optional - The spell attack bonus to use when casting the spell. If not provided, defaults to the caster's default spell attack bonus (or any attack bonus specified in the spell automation).

castingMod

optional - The spellcasting modifier to use when casting the spell. If not provided, defaults to the caster's default spellcasting modifier.

parent

optional, default None - If supplied, sets the spells created effect's parent to the given effect. This must be the name of an existing `IEffectMetaVar`. Useful for handling concentration.

Variables

No variables are exposed.

10.16 Ability Check

New in version 4.0.0.

```
{
    type: "check";
    ability: string | string[];
    contestAbility?: string | string[];
    dc?: IntExpression;
    success?: Effect[];
    fail?: Effect[];
    contestTie?: "fail" | "success" | "neither";
    adv?: -1 | 0 | 1;
}
```

An Ability Check effect forces a targeted creature to make an ability check, optionally as a contest against the caster. It must be inside a Target effect.

class Check

ability

The ability to make a check for. Must be one of or a list of the following:

```
"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"
```

If multiple skills are specified, uses the highest modifier of all the specified skills.

contestAbility

optional - Which ability of the caster's to make a contest against. Must be one of or a list of the valid skills listed above. If multiple skills are specified, uses the highest modifier of all the specified skills.

Mutually exclusive with `dc`.

`dc`

optional - An IntExpression that specifies the check's DC. If neither `dc` nor `contestAbility` is given, the check will not run either the `fail` or `success` nodes.

Mutually exclusive with `contestAbility`.

`success`

optional - A list of effects to execute on a successful check or if the **target** wins the contest. Requires the *contestAbility* or *dc* attribute to be set.

`fail`

optional - A list of effects to execute on a failed check or if the **target** loses the contest. Requires the *contestAbility* or *dc* attribute to be set.

`contestTie`

optional, default success - Which list of effects to run if the ability contest results in a tie.

`adv`

optional, default 0 - Whether the check should have advantage by default (`-1` = disadvantage, `1` = advantage, `0` = no advantage).

Variables

- `lastCheckRollTotal` (*int*) The result of the last check roll (0 if no roll was made).
- `lastCheckNaturalRoll` (*int*) The natural roll of the last check roll (e.g. `10` in `1d20 (10) + 5 = 15`; 0 if no roll was made).
- `lastCheckAbility` (*str*) The title-case full name of the rolled skill (e.g. `"Animal Handling"`, `"Arcana"`).
- `lastCheckDidPass` (`bool` or `None`) If a DC was given, whether the target succeeded the check. If a contest was specified, whether the target won the contest. `None` if no or contest given.
- `lastCheckDC` (*int* or `None`) If a DC was given, the DC of the last save roll. `None` if no DC given.

Contest Variables

- `lastContestRollTotal` (*int* or `None`) The result of the caster's contest roll; `None` if no contest was made.
- `lastContestNaturalRoll` (*int* or `None`) The natural roll of the caster's contest roll (e.g. `10` in `1d20 (10) + 5 = 15`; `None` if no contest was made).
- `lastContestAbility` (*str* or `None`) The title-case full name of the skill the caster rolled (e.g. `"Animal Handling"`, `"Arcana"`). `None` if no contest was made.
- `lastContestDidTie` (`bool`) Whether a ability contest resulted in a tie.

10.17 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)}}`).

10.18 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}`

10.19 Examples

10.19.1 Attack

A normal attack:

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

10.19.2 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]"
    }
  },
  {
    "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."
  }
]
```

10.19.3 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]"
              }
            ],
            "success": []
          }
        ],
        "miss": []
      }
    ]
  },
  {
    "type": "text",
    "text": "On a hit, a target must make a DC 12 Constitution saving throw or take 1d6␣
↪poison damage."
  }
]
```

10.19.4 Draining Attack

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

```
[
  {
    "type": "variable",
    "name": "lastDamage",
    "value": "0"
  },
  {
```

(continues on next page)

(continued from previous page)

```
    "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."
  }
]
```

10.19.5 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": [
              {
```

(continues on next page)

(continued from previous page)

```
                "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."
  }
]
```

10.19.6 Area Vampiric Drain

An effect that heals the caster for the total damage dealt to all targets:

```
[
  {
    "type": "variable",
    "name": "totalDamage",
    "value": "0"
  },
  {
    "type": "target",
    "target": "each",
    "effects": [
      {
        "type": "damage",
        "damage": "1d6 [necrotic]"
      },
      {
        "type": "variable",
        "name": "totalDamage",
        "value": "totalDamage + lastDamage"
      }
    ]
  },
  {
```

(continues on next page)

(continued from previous page)

```
    "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."
  }
]
```

10.19.7 Damage Over Time Effect

An effect that lights the target on fire, adding two buttons on their turn to take the fire damage and douse themselves.

```
[
  {
    "type": "target",
    "target": "each",
    "effects": [
      {
        "type": "ieffect2",
        "name": "Burning",
        "buttons": [
          {
            "label": "Burning",
            "verb": "is on fire",
            "style": "4",
            "automation": [
              {
                "type": "target",
                "target": "self",
                "effects": [
                  {
                    "type": "damage",
                    "damage": "1d6 [fire]"
                  }
                ]
              },
              {
                "type": "text",
                "text": "At the start of each of the target's turns, the target takes␣
↪1d6 fire damage."
              }
            ]
          },
```

(continues on next page)

(continued from previous page)

```
          {
            "label": "Douse",
            "verb": "puts themself out",
            "automation": [
              {
                "type": "remove_ieffect"
              },
              {
                "type": "text",
                "text": "The target can use an action to put themselves out."
              }
            ]
          }
        ]
      }
    ]
  }
]
```

10.20 Custom Attack Structure

```
{
    _v: 2;
    name: string;
    automation: Effect[];
    verb?: string;
    proper?: boolean;
    criton?: number;
    phrase?: string;
    thumb?: string;
    extra_crit_damage?: string;
    activation_type?: number;
}
```

In order to use Automation, it needs to be contained within a custom attack or spell. We recommend building these on the Avrae Dashboard, but if you wish to write a custom attack by hand, the structure is documented here.

Hand-written custom attacks may be written in JSON or YAML and imported using the `!a import` command.

class AttackModel

_v

This must always be set to `2`.

name

The name of the attack.

automation

The automation of the attack: a list of effects (documented above).

verb

optional, default "attacks with" - The verb to use in attack title displays.

proper

optional, default false - Whether or not the attack's name is a proper noun. Affects title displays.

criton

optional - The natural roll (or higher) this attack should crit on. For example, `criton: 18` would cause this attack to crit on a natural roll of 18, 19, or 20.

phrase

optional - A short snippet of flavor text to display when this attack is used.

thumb

optional - A URL to an image to display in a thumbnail when this attack is used.

extra_crit_damage

optional - How much extra damage to deal when this attack crits, in addition to normal crit rules such as doubling damage dice. For example, if this attack normally deals 1d6 damage with `extra_crit_damage: "1d8"`, it will deal 2d6 + 1d8 damage on a crit.

activation_type

optional - What action type to display this attack as in an action list (such as `!a list`).

```
ACTION = 1
NO_ACTION = 2
BONUS_ACTION = 3
REACTION = 4
MINUTE = 6
HOUR = 7
SPECIAL = 8
LEGENDARY = 9
MYTHIC = 10
LAIR = 11
```

10.21 Specifying Class Feature DC Bonuses

New in version 4.1.0.

Many official class automations let you specify a DC bonus that is added to the class feature's DC. For example, to add a bonus to all of your Fighter's Battlemaster Maneuvers, you can set a `FighterDCBonus` cvar and add it to the DC of all of your maneuvers.

For more details on using this, see *Specifying Class Feature DC Bonuses*

To account for this in your automations, use the *Set Variable* node, with a value of `XDCBonus` and an onError of 0.

```
{
  "type": "variable",
  "name": "BloodHunterDCBonus",
  "value": "BloodHunterDCBonus",
  "onError": "0"
}
```

Then, when you set your save DC's in that automation, add `+XDCBonus` to the DC total.

```
{
    "type": "save",
    "stat": "str",
    "dc": "8+proficiencyBonus+intelligenceMod+BloodHunterDCBonus",
    "fail": [],
    "success": []
}
```

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

A

B

C

D

I

J

L

M

N

O

P

R

S

T

U

V

W