Tagscript-Docs

Release 0.1

_Leg3ndary

Apr 19, 2022

INFO

Attention: This is not affiliated in any way with Botlabs or Carl-bot.

Attention: Really would like to thank readthedocs.org for making this possible, thank you!

This is an unofficial guide to Carl-bot's tagscript, started by _Leg3ndary#5759, you may view all contributors *here.*

Note: All of this documentation contains custom tagscript syntax highlighting through tagscript-ansi

```
{=(L1):{lower:{1}}}
{=(A2+):{replace(|, - ):{args(2+)}}}
{=(error):You must follow the `afk` command with either `on` or `off`.}
{=(on):add}
{=(off):del}
{=(template):c:ar {{L1}} {user(id)}>{if({L1}==on): {if({list(0):{join(,):{A2+}}}!=):{A2+}
↪|{replace(|, - ):{user}} is afk right now, send them a PM or wait for them to return.}}
↪}
{=(sel):{if({contains({L1}):on off}==false):error|{template}}}
{override}{{sel}}

Raffael#1372's AFK Tag
```

Tip: If you just want a quick refresher on blocks, this tag will contains everything you need to know: ?tse <block>

Note: This documentation is still under development.

CHAPTER

ONE

CONTRIBUTING

If you believe something here is incorrect or would like to add something else, submit a pull request on Github.

All help is appreciated! :)

Important: Please include your `username#discriminator` as well as your id `123456789012345` as you should have some experience with tagscript.

Note: If you do decide to contribute, make sure to check out *Credits* so you can be added!

1.1 A Basic Guide to Contributing

These docs use ReStructuredText or RST, it's a fairly simple language that you can learn here RST Docs.

You could also take a look at the readthedocs sphinx theme for more info.

Finally tagscript parsing is here, though there's not much to know.

Note: If you want to easily create tagscript you can do so by doing the following

```
.. tagscript::

    {=(Tagscript):Can go here}
```

This will generate the following

```
{=(Tagscript):Can go here}
```

CHAPTER TWO

CREDITS

Note: This page isn’t complete

Massive thanks to anyone who’s contributed to tagscript in Carl-bot’s Support Server, I’m working on figuring out why I can’t add you with profile pictures but you will be added.

CHAPTER THREE

LIMITS

There are many limits in tagscript, and some more when you use Carl.

3.1 Loops

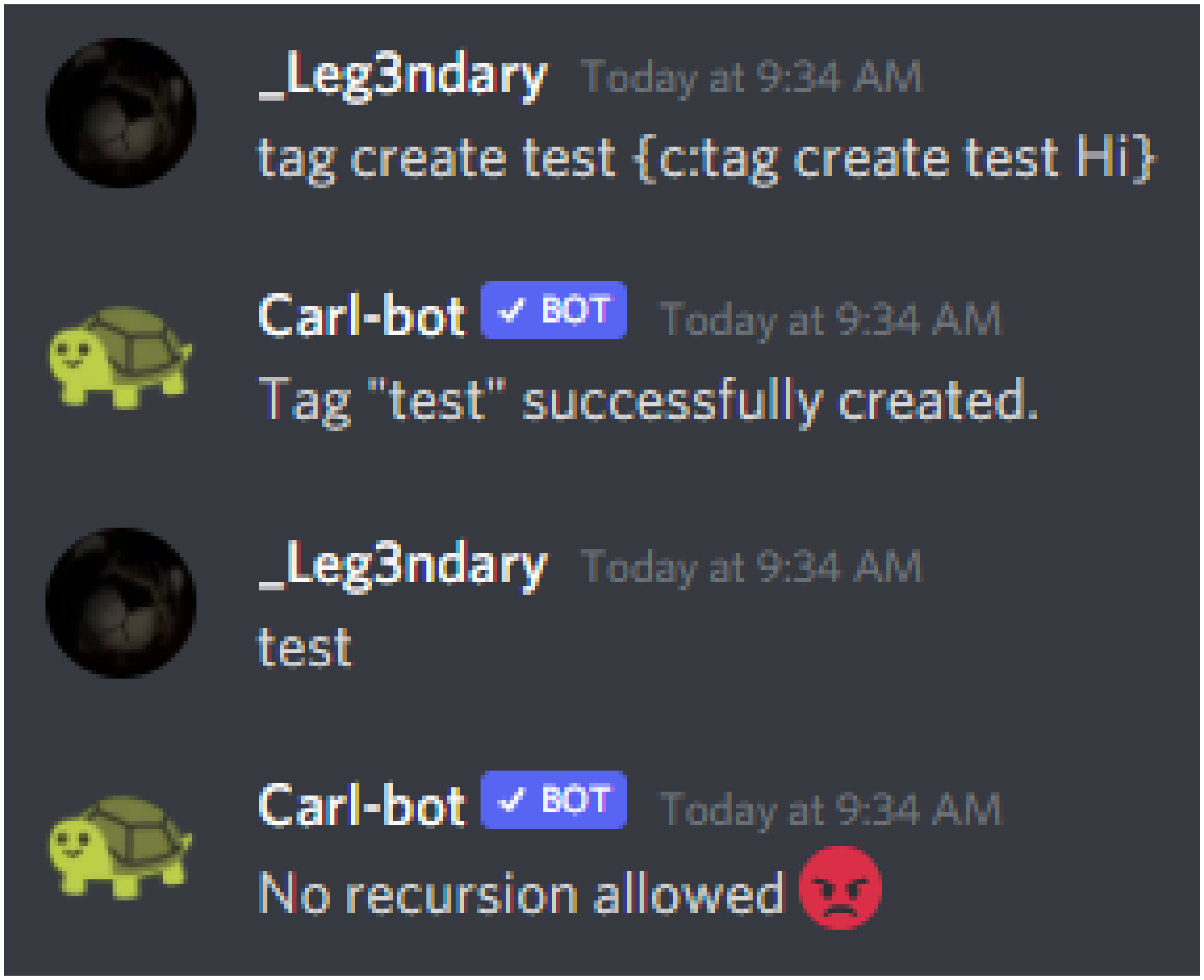
You will not have access to loops in any way, there are a few exceptions, but in general if you want to loop through values and edit them, that won't be possible.

Examples of blocks that you can use that have loop-like features:

```
{replace}
{join}
```

3.2 Tag Recursion

If you're looking to save data through tag recursion by using `tag add` sadly this will not work, you'll get the following message


_Leg3ndary Today at 9:34 AM
tag create test {c:tag create test Hi}

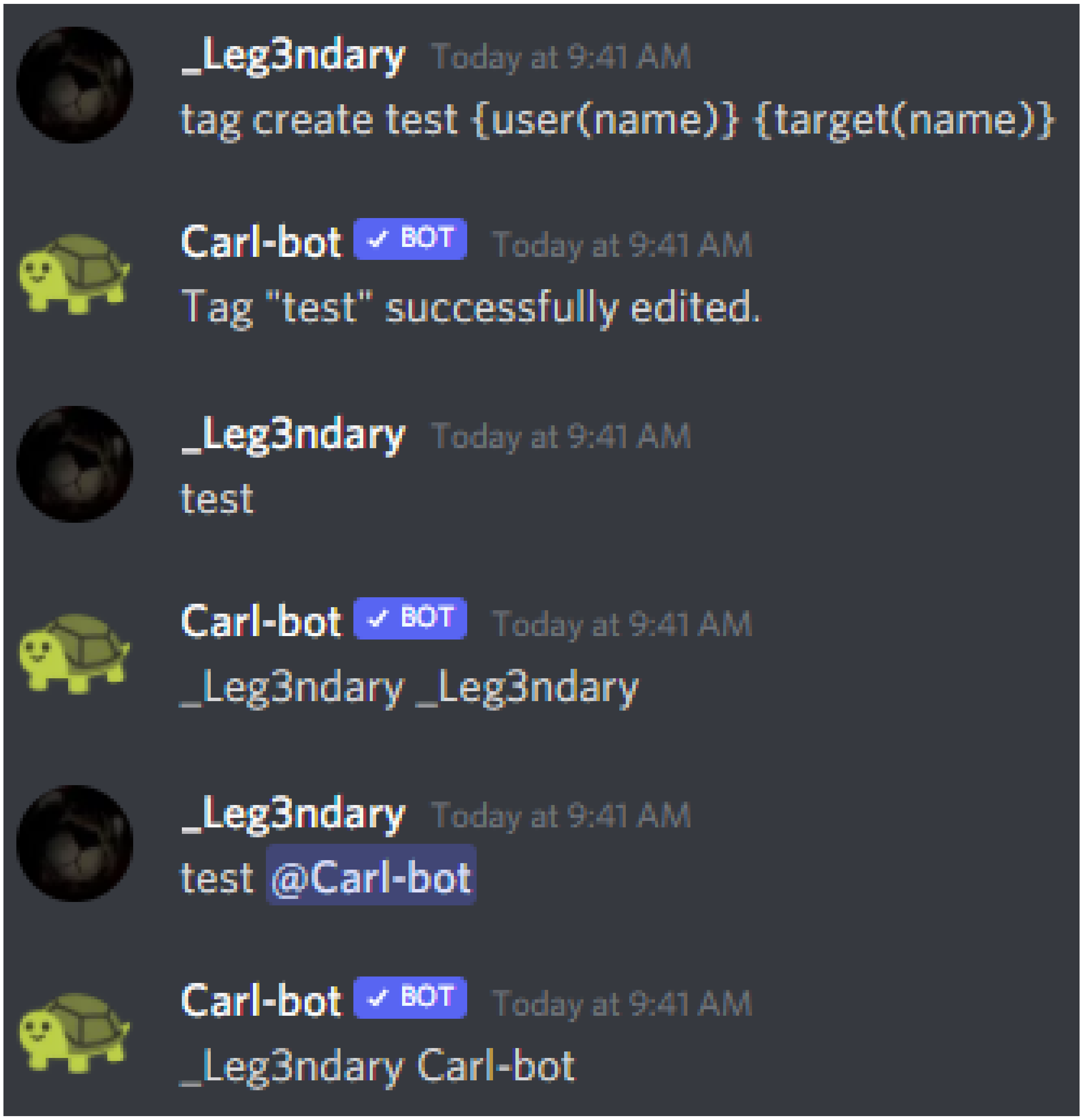
Carl-bot ✓ BOT Today at 9:34 AM
Tag "test" successfully created.

_Leg3ndary Today at 9:34 AM
test

Carl-bot ✓ BOT Today at 9:34 AM
No recursion allowed 😡

3.3 Accessing Other Users Data

You will only be able to access 2 different users when using tags, and you must ping at least one of them.



_Leg3ndary Today at 9:41 AM
tag create test {user(name)} {target(name)}

Carl-bot ✓ BOT Today at 9:41 AM
Tag "test" successfully edited.

_Leg3ndary Today at 9:41 AM
test

Carl-bot ✓ BOT Today at 9:41 AM
_Leg3ndary _Leg3ndary

_Leg3ndary Today at 9:41 AM
test @Carl-bot

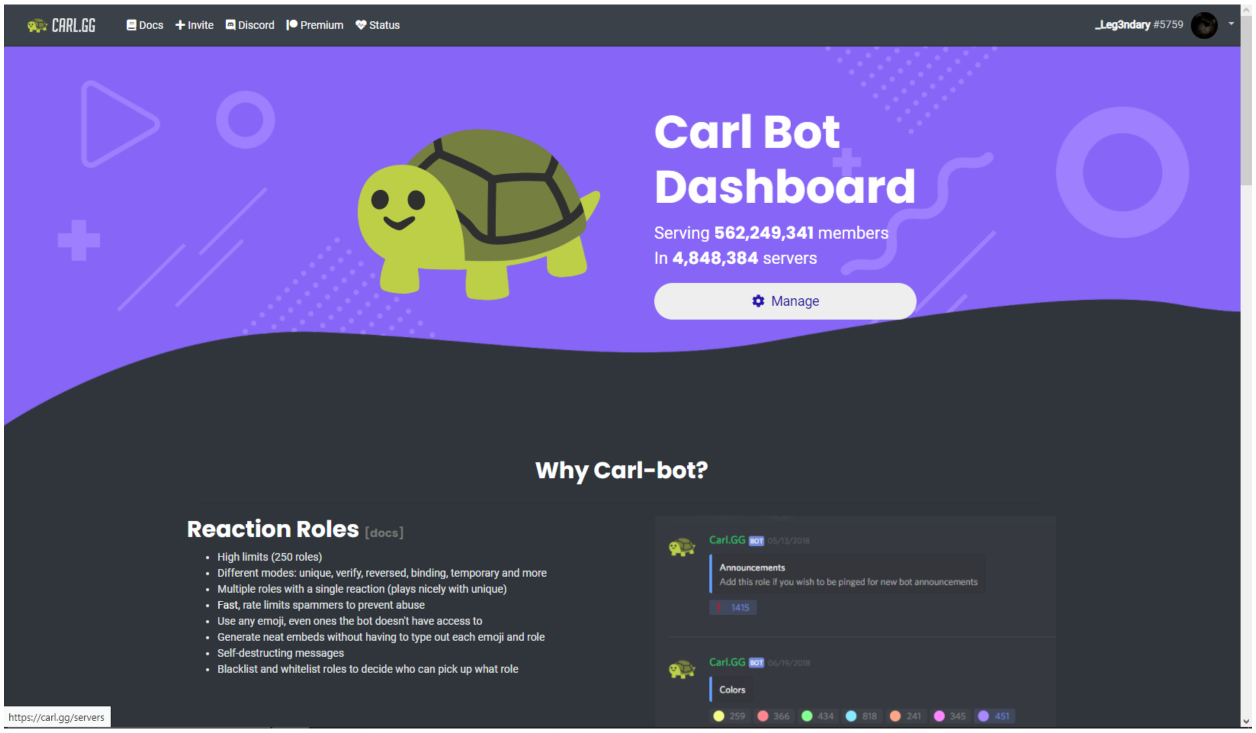
Carl-bot ✓ BOT Today at 9:41 AM
_Leg3ndary Carl-bot

CHAPTER

FOUR

CREATING TAGS

There are 2 different ways you can create tags, through the Dashboard or with commands



Carl Bot Dashboard

Serving **562,249,341** members

In **4,848,384** servers

Why Carl-bot?

Reaction Roles [docs]

- High limits (250 roles)
- Different modes: unique, verify, reversed, binding, temporary and more
- Multiple roles with a single reaction (plays nicely with unique)
- Fast, rate limits spammers to prevent abuse
- Use any emoji, even ones the bot doesn't have access to
- Generate neat embeds without having to type out each emoji and role
- Self-destructing messages
- Blacklist and whitelist roles to decide who can pick up what role

Carl.GG BOT 05/13/2018

Announcements

Add this role if you wish to be pinged for new bot announcements

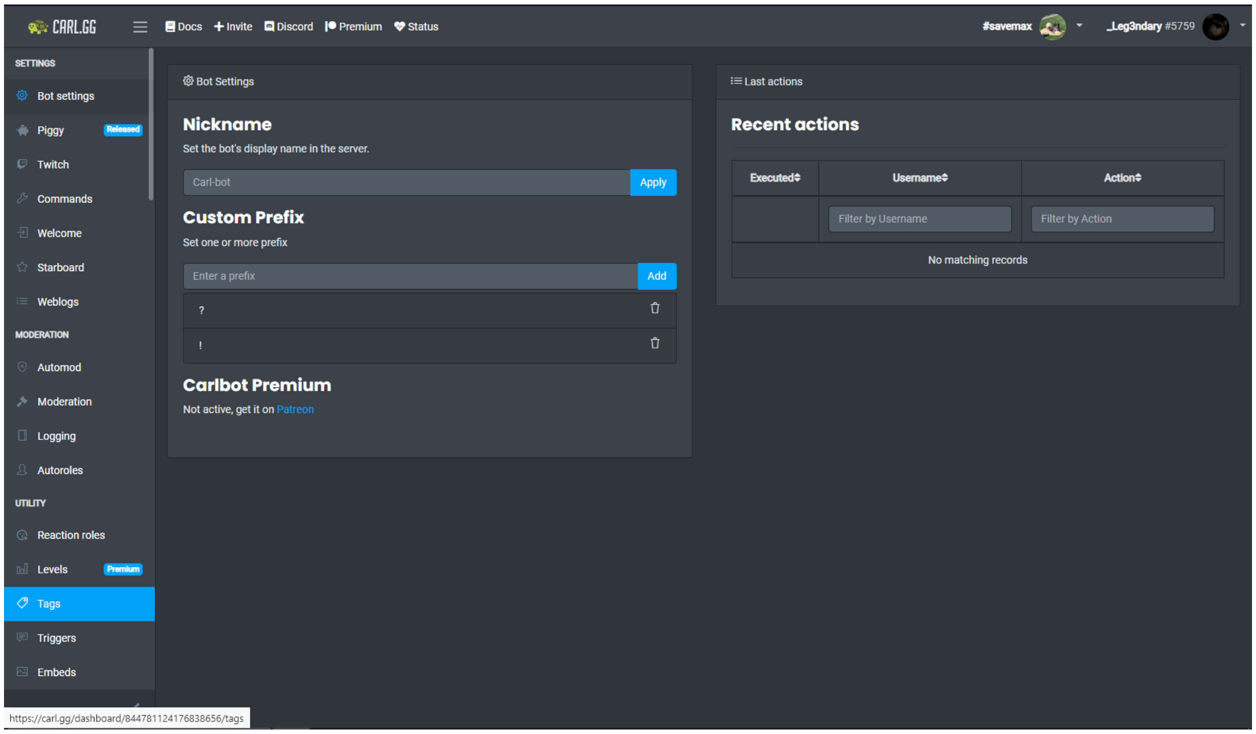
1415

Carl.GG BOT 06/19/2018

Colors

259 366 434 818 241 345 451

After selecting a server, head into the **Tags** section



Bot Settings

Nickname

Set the bot's display name in the server.

Carl-bot Apply

Custom Prefix

Set one or more prefix

Enter a prefix Add

?

!

Carlbot Premium

Not active, get it on Patreon

Last actions

Recent actions

Executed	Username	Action
	Filter by Username	Filter by Action
No matching records		

CHAPTER

FIVE

ANATOMY OF BLOCKS

When we refer to blocks we mean anything that contains both an opening `{` and a closing `}`.

Warning: Carl-bot will always evaluate brackets so because of this behaviour, you may not have `{` or `}` in any block.

There is no way to bypass this while writing your code.

5.1 Block Names

Block names are what's inside the curly brackets, these will always be the first thing after `{` and is mandatory for every block.

Note: An exception is made for variables in which you can literally define variables to be "blank" `{=():}`, if you don't understand this, don't worry, it's not important right now.

Some examples include:

```
{user}
{command}
{let}
```

The name of the block will determine how the parameters and payload are evaluated.

5.2 Parameters and Payloads

5.2.1 Parameters

Parameters will be defined straight after the block name `{block(PARAMETERS)}`, parameters will usually let us alter the payload depending on what we put in it.

```
{user(PARAMETERS)}
{command(PARAMETERS)}
{let(PARAMETERS)}
```

Warning: You also may not have any `(` or `)` in parameters.

5.2.2 Payloads

Payloads will also be defined after the command name using a `:`, unless parameters were added, in that case, it will go straight after parameters instead. Payloads are the text that we want to alter/use.

```
{user(PARAMETERS):PAYLOAD}
{command:PAYLOAD}
```

Note: Payloads may contain `:`

Note: When working with conditional blocks such as `if`, `or`, or `and`, you may not be able to use vertical pipes `|`, as it can break the *else* condition.

Note: Remember that payloads and parameters can be optional, meaning all of these are possible with the right blocks.

```
{block(PARAMETERS)}
{block:PAYLOAD}
{block(PARAMETERS):PAYLOAD}
```

5.3 Examples

```
The blocks' parameter in this case is "avatar"
{user(avatar)}

The blocks' payload in this case is "lock server"
{command:lock server}

The blocks' parameter here is "tagscript", while the payload is "cool"
{let(tagscript):cool}
```

Important: It's strongly advised that you familiarize yourself with basic anatomy to avoid common errors, this will also later be essential to understand more advanced concepts such as blanks and switches.

CHAPTER

SIX

DEFAULT VARIABLES

Defualt variables are variables that every tag will have defined.

6.1 Unix

```
{unix}
```

Returns the current unix time

Often used in conjunction with strf and timedelta blocks

Note: To use this outside of tags, add this to the top of your code `{=(unix):{strf:%s}}`

Warning: This is exclusive to tags.

6.2 Uses

```
{uses}
```

Returns how many times your tag was used

Warning: This is exclusive to tags.

Warning: This increments even if the tag fails to output/work, the only way to reset this is to delete and re-import the tag.

6.3 Mention

```
{mention}
```

Mentions the user who used the tag

Note: This is the same as `{user(mention)}`

6.4 Args/Message

Important: This is probably the most used and important block in tagscript, it's what allows you to access what users put after an invocation.

In addition you will often need to parse this block, so it would be wise to check out *Parsing*.

```
{args}
{message}
```

What is after a tag invocation:

```
?foo bar baz

{args} will output bar baz
```

Note: If used in a trigger, this block will also contain the trigger invocation.

6.4.1 Digit Shorthands

```
{1}
{2}
{3} etc.
```

The main difference between args and message is how message has digit shorthands.

```
{args(1)} is equivalent to {1}
{args(2)} is equvalent to {2}
```

However digit shorthands are based upon the message variable, meaning if you change it, by redefining it, digit shorthands will now be based on that instead.

Note: If you don't understand digit shorthands, don't worry! They aren't really used and it's much more common to see people use `{args(1)}` over `{1}`!

CHAPTER

SEVEN

DISCORD OBJECTS

Discord related default variables

7.1 User/Target

```
{user}
{target}

{user(name)}
{target(avatar)}
```

One of the most used blocks is the target block, it allows you to access other users data.

`user` is the user who invoked the tag, while `target` is the first user that was pinged in the commands arguments. If no one is pinged, `target` is equivalent to `user`.

Note: It's common to check if someone was pinged in the command, to do this you compare user against target

```
{if({user(id)}=={target(id)}):You need to ping someone|You pinged {target}}
```

7.1.1 Parameters/Properties

Parameters can be specified within parenthesis after the block name: `{user(PARAMETER)}` or `{user(PARAMETER)}`. When used without a parameter (`{user}` or `{target}`), the block will output the nickname of the user/target.

Parameter	Output
avatar	A link to the user's avatar
icon	Same as avatar
id	The user's ID
mention	@'s/Mentions the user
created_at	Date and time that the account was created in the form of `yyyy-mm-dd HH:MM:SS`
joined_at	Date and time that the user joined this server in the form of `yyyy-mm-dd HH:MM:SS`
color	The user's hexadecimal role color with the leading # included
name	The user's discord username
proper	The user's username + discriminator/tag, example `_Leg3ndary#5759`
roleids	A list of every role (id) the user has from lowest to highest separated by a space
position	The user's position in the role hierarchy, starts with 0 for @everyone and increases by 1 for each role in the server

7.2 Server

```
{server}

{server(roles)}
```

Contains details about the server

7.2.1 Parameters

Parameters can be specified within parenthesis after the block name: `{server(PARAMETER)}`. When used without a parameter (`{server}`), the block will output the server's name.

Warning: Some of these are deprecated.

Parameter	Output
icon	A link to the server's icon
id	The server's ID
owner	The username + discriminator of the server owner
random	The username + discriminator of a random member
members	The number of members in the server
roles	The number of roles in the server
channels	The number of channels in the server
created_at	When the server was created in the format `yyyy-mm-dd HH:MM:SS`
[BROKEN] humans	The number of humans in the server. Currently returns the same as members.
[DEPRECATED] randomonline	The username + discriminator of a random online member
[DEPRECATED] randomoffline	The username + discriminator of a random offline member
[DEPRECATED] bots	The number of bots in the server

7.3 Channel

```
{channel}

{channel(topic)}
```

Contains details about the channel

7.3.1 Parameters

Parameters can be specified within parenthesis after the block name: `{channel(PARAMETER)}`. When used without a parameter (`{channel}`), the block will output the channel's name.

Parameter	Output
id	The channel's ID
topic	The channel's topic
slowmode	The channel's slowmode delay in seconds
position	The channel's position, in the order of which channels were created, 0 being the first
mention	Clickable link to the channel

CHAPTER EIGHT

META

Meta blocks are blocks that will change a tags overall behaviour, this includes changing where it replies, deleting the invocation, and interacting with users.

8.1 Delete

```
{delete}
{del}
```

Deletes the message which invoked the commmand

8.2 Silent

```
{silent}
{silence}
```

Silences any command block outputs

8.3 Override

```
{override}
```

Overrides any permissions needed to run a command

Danger: Use this with caution.

Warning: This still respects hierarchy, for example if you're trying to add a role to yourself, using carl, or someone who has a higher role then you. This will not work.

8.4 Direct Message

```
{dm}
```

This will instead send the output to a users direct messages (DMS)

Note: This won't ever be able to send anything to another users dms. The bot will only dm you if you use a tag with this block in it.

8.5 Redirect

```
{redirect:channel_id}
```

Redirect will redirect the output to whatever channel id you provide

Warning: The user must have the `send_messages` permission for this to work, or you can add an `override` block

8.6 Require/Blacklist

Important: You can use technically use the role/channel name though it is heavily discouraged, if that name ever changes, the tag will cease to work properly, while using ids will stay the same forever.

```
{require(Optional Error Message):Required Roles, Channels}
{blacklist(Optional Error Message):Blacklisted Roles, Channels}

{require(You aren't a moderator, or you aren't using this in the right␣
↪channel):209797471608635392,465563733981265921}
{blacklist(Muted users aren't allowed to use this command, if you aren't muted, use #bot-
↪commands):469237398279159818,456625369974308866}
```

Require is the easiest way to require a user to have a role, or use it in a specific channel.

Blacklist is also an easy way for blacklisting certain channels and or roles from using tags.

Require Blocks: Separate the channels or roles by a `,` with no spaces inbetween, and as long as they have one of the roles, and one of the channels (If you have both) they will be able to use the tag.

Blacklist Blocks: Also separate the channels or roles by a `,` with no spaces inbetween, if they have any of the roles or are using it in any of the channels, the tag will output your error message

8.6.1 Parameters

```
{require(ERROR MESSAGE):209797471608635392}

{blacklist(ERROR MESSAGE):469237398279159818}
```

Note: This is optional, you can simply do `{require:209797471608635392}` or `{blacklist:469237398279159818}`

The error message that will output when the user doesn't have one of the roles or isn't using it in a channel listed.

The error message that will output when the user is using the tag when they have a role or are using it in a blacklisted channel.

Warning: If you do choose to omit the parameter for either blacklist or require, carl will react with a emoji instead, to avoid this simply make the error message a space like so

`{require( ):209797471608635392}` or `{blacklist( ):469237398279159818}`

Warning: This is exclusive to tags

8.6.2 Payload

```
{require(You aren't a moderator!):ID LIST}

{require(You can't use this command here!):ID LIST}
```

A list of role or channel ids separated by `,` with no spaces inbetween.

As long as the user using the tag has one of the role ids, and one of the channel ids (If you have both) they will be able to use the tag.

Important: When using blacklist, you can blacklist the server id to automatically break the tag if you want, this is most often used when you want to prevent an embed from posting

8.7 React(u)

```
{react: :turtle: :robot:}
{reactu: :turtle: :robot:}
```

React blocks will react to what carl outputs, while reactu blocks will react to the tags invocation.

Attention: In tags you're limited to 1 reaction without premium, and 5 with premium

In triggers you're limited to 3 reactions without premium, and 5 with premium *I believe

8.7.1 Payload

```
{react:EMOJI LIST}
{reactu:EMOJI LIST}
```

The emoji list should be separated by spaces, for custom discord emojis, send a \ in front of it and send it to a channel, use what you then see.

CHAPTER

NINE

COMMANDS

Caution: Want to stop a command block from conditionally running? Remember to use *blanks*!

```
{c:Command}
{cmd:role add _Leg3ndary Red}
{command:ban Carl-bot Useless bot}
```

Command blocks execute Carl's commands, it's what allows you to add roles, change nicknames, etc.

Note: Every server is limited to a maximum of **one** command block, an exception is made for premium servers who are allowed up to three.

9.1 Payload

```
{c:COMMAND YOU WANT TO RUN}
```

Tip: If you want to create an alias for an existing command, it's as simple as creating a tag with the alias as it's name and adding a command block with args

```
{c:COMMAND NAME {args}}

{c:mute {args}}

{c:ban {args}}
```

Warning: Tags aren't able to run reactionrole commands nor can they run tag commands, eg tag add, or rr add

CHAPTER

TEN

CONTROL

Control blocks “control” what happens based on the condition we give it

Consider the following:

```
If I, am hungry, I will eat, if not, I will drink some water.

To convert this into tagscript we first create an if statement

{if}

Now we create our condition

{if(I==hungry)}

Now we add the payload or what will happen

{if(I==hungry):Eat}

Finally we add the else

{if(I==hungry):Eat|Drink}
```

Note: This just pseudo code! If you run this, it will always say Drink Water as I is not the same as hungry!

10.1 Boolean Operators (Parameters)

```
{if(CONDITION):true|false}

{if(carl==carl):true|false} -> true

{if(carl!=carl):true|false} -> false

{if(5>1):true|false} -> true

{if(5<1):true|false} -> false
```

This is how we determine what to do, in this case we just print out true or false

==

If the left side is exactly the same as the right side, space and case sensitive

!=

If the left side is different from the right side, space and case sensitive

>=

Greater than or equal too

<=

Lesser than or equal too

>

Greater than

<

Lesser Than

Tip: An extremely common question, is how we check if a user was pinged!

We can easily check this by comparing the user id, to target id

```
{if({user(id)}=={target(id)}):You need to ping someone!|You pinged {target}}
```

10.2 Then/Else (Payload)

```
{if({user(id)}==235148962103951360):THEN|ELSE}

{if({uses}>10):This command has been used more then 10 times|This command has only been␣
↪used {uses} times!}

{and({target}==Carl-bot|{target(id)}!=235148962103951360):How dare you impersonate me!}
```

The payload for conditional blocks can either be a then without an else, or both, you separate these with a pipe `|`.

10.3 If

```
{if(CONDITION):THEN|ELSE}
```

The simplest of conditional blocks, checks a singular condition.

10.4 Any/Or

```
{any(CONDITION|CONDITION|CONDITION):THEN|ELSE}
```

If you want to check if any condition out of whatever you provide are true, you can use an any block, just separate every condition with a `|`.

10.5 All/And

```
{and(CONDITION|CONDITION|CONDITION):THEN|ELSE}
```

Nearly identical to the any block, this block just checks if every condition you provide is true.

10.6 Break/Shortcircuit

```
{break(CONDITION):THEN}
```

When used, if the condition given is true, the tags text output will only be whatever you put as the payload.

Danger: This will not prevent command blocks from running or the embed from the embed builder from sending.

CHAPTER ELEVEN

DATA STORAGE/VARIABLES

Note: Tags cannot store data between invocations through regular means, you have other options though! See here.

```
{=(VARIABLE NAME):VARIABLE CONTENT}

{=(tag_name):afk}
{tag_name} -> afk

{let(new nick):Default User}
{new nick} -> Default User

{var(formulaEQ):5+6(x-1)}
{formulaEQ} -> 5+6(x-1)

{assign(userGoal):1,000,000 Members!}
{userGoal} -> 1,000,000 Members!
```

An extremely important block, the variable block allows you to save data per command invocation and use/alter it.

To define a variable, you just need to give it a name, and content, which may be empty.

After that, to call it, all you have to do is put brackets around the variables name.

Warning: You cannot name variables, names of existing blocks, for example, you cannot name a variable `user`, as that already exists (See Discord.rst)

CHAPTER

TWELVE

PARSING

Parsing is an extremely important part of tagscript, and there are a few ways in which you can parse text.

Note: Parsing can be done with basically anything, though you shouldn't try to parse datetimes using methods below, that's what *strf blocks* are for.

12.1 Variable Parsing

```
{=(test_var):Carl-bot is the best bot! His favorite food is subway!}

{test_var} -> Calling the variable
{Variable Name(Elements):Delimiter}

{test_var(1)} -> Calling the first "word" (Basically setting the delimiter to a space)
The above would return Carl-bot

{test_var(1):!} -> Calling the first part of the variable up to the first "!"
The above would return Carl-bot is the best bot
```

One big function of variables is the ability to parse them, you can specify any number of elements and can also change the delimiter.

Note: This also works with {args} and {message}.

12.1.1 Elements

```
{=(test_var):Carl-bot is the best bot! His favorite food is subway!}

{test_var(ELEMENT)}

{test_var(4)} -> Calling the fourth element
The above would return best

{test_var(0)} {test_var(-2)} -> Calling the last element and the third last element.
The above would return subway! food
```

(continues on next page)

(continued from previous page)

```
{test_var(+3)} | {test_var(7+)} -> Calling everything up to the third element and␣
↪everything from the seventh onward
The above would return Carl-bot is the | favorite food is subway!
```

Elements are how you control how many elements you return.

12.1.2 Delimiters

```
{=(test_var):Carl-bot is the best bot! His favorite food is subway!}

{test_var(ELEMENT):DELIMITER}

{test_var(2):!} -> Calling the second argument when test_var is split by ! instead of␣
↪spaces
The above would return His favorite food is subway!
```

When a delimiter isn't specified, tagscript automatically assumes you mean a space, or basically `{args(5)}` is the same as `{args(5):  }`.

Delimiters on their own don't do anything, you need to specify an element for this to change anything.

Note: This is most often used to get certain parts of variables, an example being a urls domain name.

Knowing that urls are formatted like so: `https://readthedocs.org/dashboard/` We can first parse everything after the `//` and then everything before the first `/`.

```
{=(url):https://readthedocs.org/dashboard/}
{=(url):{url(2)://}}
{=(url):{url(1):/}}
{url}
```

Warning: Just to reiterate `{test_var:DELIMITER}` won't do anything

12.2 List & Cycle

```
{list(INDEX):elem,elem2,elem3,elem4}
{list(INDEX):elem~elem2~elem3~elem4}

{cycle(INDEX):elem,elem2,elem3,elem4}
{cycle(INDEX):elem~elem2~elem3~elem4}
```

List blocks will return the element at whatever index you specify. If you specify an index that's out of bounds, the block will return nothing.

Cycle blocks will work the same, however when specifying an index that's out of bounds, the block will *cycle* back.

When separating elements you may use `,` or `~`, however if you have both, the tilde will take precedence.

12.2.1 Index

```
{list(-1):elem~elem2~elem3~elem4} -> elem4
{list(1):elem~elem2~elem3~elem4} -> elem2

{cycle(5):elem~elem2~elem3~elem4} -> elem2
{cycle(-6):elem~elem2~elem3~elem4} -> elem3
```

You may parse this similarily to regular parsing, however you may only parse one element at a time.

You also may use negative numbers to go backward.

Attention: Indexes start at 0, meaning the first element will have index 0, the second, 1 etc. etc.

12.3 Index

```
{index(ELEMENT):elem~elem2~elem3~elem4}
```

Index is quite straightforward, it will simply index the element and return its position.

12.3.1 Element

```
{index(elem2):elem~elem2~elem3~elem4} -> 1

{index(elem5):elem~elem2~elem3~elem4} -> -1
```

Note that this block will always return the first found instance regardless of how many times it's found in the string, in addition if the element isn't found, the block will return -1.

Attention: Indexes start at 0, meaning the first element will have index 0, the second, 1 etc. etc.

12.4 Membership Testing (In & Contains)

```
{in(STRING):TEXT}

{contains(ELEMENT):LIST}
```

These blocks test if a list or piece of text has a string or element in it.

In is the more powerful of the two, it will check if the string is in the text regardless of where it is while contains must have spaces around the given text.

This will return a bool value of true or false.

```
{in(cool):Carl-bot is cool!} -> true
{contains(cool):Carl-bot is cool!} -> false

```

(continues on next page)

(continued from previous page)

```
{in(efg):abcdefghijklmnop} -> true
{contains(efg):abcdefghijklmnop} -> false

{in(Carl):Carl bot} -> true
{in(carl):Carl bot} -> true
{contains(Carl):Carl bot} -> true
{contains(carl):Carl bot} -> false
```

Attention: Everything in tagscript is case-sensitive, this includes contains and in blocks, you can use the lower or upper block to null this though.

EMBED

```
{embed(EMBED PARAMETER):VALUE}

{embed(title):Embed Title}
{embed(timestamp):now}
```

Embed blocks are quite simple, they simply change the value in the embed sent via the embed builder.

Attention: You **MUST** have a value in the embed builder, anywhere, if the embed builder is empty, the tag will not send the embed.

EMBED PARAMETER	Output
title	Sets the embed title
url	Sets the embed url for the title
description	Sets the embeds description
color	Sets the embeds hex color, must include #
timestamp	Sets the embeds timestamp, the only possible payload for this is now.

CHAPTER FOURTEEN

MANIPULATION

14.1 Ordinal Abbreviation

```
{ord:INT}

{ord:1} returns 1st
{ord:22} returns 22nd
{ord:37} returns 3th
{ord:456} returns 456th
```

The ordinal abbreviation block returns exactly what you'd expect it to, the ordinal abbreviation of the number.

Note: If you don't supply a number or the number has commas/decimals, this block will return `<ord error>`.

14.2 Case Blocks

```
{lower:Whoozard is a Wizard} -> whoozard is a wizard
{upper:carl-bot best bot} -> CARL-BOT BEST BOT
```

Case blocks also do exactly what'd you'd expect them too, they either upper or lower what you want, this is often used to make inputs case insensitive.

14.3 Text Replacement

```
{join(STRING):TEXT}
{replace(FIRST STRING,SECOND STRING):TEXT}
{urlencode(SPACE ENCODING):URL}

{join(_):Carl-bot is cool} -> Carl-bot_is_cool
{replace(, ):Carl-bot is cool} -> C a r l - b o t   i s   c o o l
{replace(l-bot,w-bot):Carl-bot is cool} -> Carw-bot is cool
{urlencode:Hey there, how are you?} -> Hey%20there%2C%20how%20are%20you%3F
{urlencode(+):Hey there, how are you?} -> Hey+there%2C+how+are+you%3F
```

Text replacement blocks also do what you would expect them too.

Join will join the strings spaces with whatever string you provide it with, this means it's the equivalent of `{replace( ,STRING):TEXT}`.

Replace is similar to join but you separate what you want to replace and what it will be replaced with, with a comma. This can be used to space variables `{replace(, ):TEXT}`.

Urlencode will encode any text (Usually URLs obviously) into percent form. The only possible value for SPACE ENCODING is +.

Note: Since you must separate what you want to replace with comma, you aren't able to replace commas.

CHAPTER FIFTEEN

MATH

```
{m:EQUATION}

{math:9+10} -> 19
{calc:365-5} -> 360
{+:14/7} -> 2
{m:12*10} -> 120
```

The math block also works as you'd expect it to, the block itself **only** takes a payload which will be calculated in the correct order of operations.

15.1 Basic Functions and Operators

Function/Operator	Explanation
x+y	Addition
x-y	Subtraction
x*y	Multiplication
x/y	Division
x%y	Modulo
x^y	Exponent
abs(x)	Absolute value
round(x)	Rounds to the nearest whole number
trunc(x)	Truncation

15.2 Advanced Functions, Operators and Vars

Function/Operator/Var	Explanation
sin(x)	Sine (radians)
cos(x)	Cosine (radians)
tan(x)	Tangent (radians)
exp(x)	Eulers number to the x power
sgn()	Returns the sign of a number, 1 positive, -1 negative, 0 0
log(x)	Logarithm
ln(x)	Natural Logarithm
log2(x)	Logarithm with base 2
pi/PI	Will be replaced with pi
e/E	Will be replaced with Euler's number

Warning: When calculating long equations carl will occasionally hiccup and just output the entire equation, there's nothing you can do about this.

Try it for yourself

CHAPTER

SIXTEEN

RANDOM NUMBER GENERATORS

16.1 Random

```
{random(OPTIONAL SEED):List,of,elements}
{rand(OPTIONAL SEED):List~of~elements}
{#(OPTIONAL SEED):4|Weighted,2|list,of,3|elements}
```

Random blocks just pick a random choice out of a list you provide, you can also provide optional seeds and weight elements to prevent repetition.

16.1.1 Optional Seeding

Seed values are completely optional, however when you use a certain seed value with the same list of elements, the same element will always be chosen.

You can think of this to almost be a key that works almost like a cycle block.

16.1.2 Weighting

```
{#:4|Carl,2|bot} == {#:Carl,Carl,Carl,Carl,bot,bot}

{#:4|Lose,Win}
```

Weighting is just a simple way to add more elements without typing as much, keep in mind you don't need to provide weighting for every value, the last example shows an example of having a 1 in 5 chance of winning.

16.2 Range

```
{range(OPTIONAL SEED):LOWER-HIGHER}
{rangef(OPTIONAL SEED):LOWER-HIGHER}

{range:1-100}
{rangef:0-1}
```

Range blocks generate random numbers between the 2 numbers given (inclusive) while rangef blocks generate random numbers with a single decimal (also inclusive).

Seeds may also be provided and work exactly the same as random blocks.

16.3 5050 Blocks

```
{5050:OPTION}
{50:OPTION}
{?:OPTION}
```

Has a 5050 chance of choosing said option, or nothing at all.

CHAPTER

SEVENTEEN

TIME

17.1 STRF

```
{strf(OPTIONAL DATETIME OR UNIX):STRF FORMAT CODE}

December 31st 1999 was a {strf(1999-12-31 23.59.59):%A} -> December 31st 1999 was a␣
↪Friday
The current time is {strf:%-I:%M %p} -> The current time is followed by whatever the␣
↪current 12h clock is for UTC, eg 2:19 AM
Your account was created on {strf({user(created_at)}: %x} -> Your account was created on␣
↪2015/12/24 (For Carl)
```

When dealing with time we often want it to be outputted in different ways, strf does exactly that.

You can either supply a datetime or unix value, if none are supplied, the engine assumes you want the current time.

Note: For a full list of STRF codes, check here

Note: Some useful codes

```
%Y-%m-%d %H:%M:%S & the short version %F %T are the DateTime format in strf codes, which␣
↪can be useful for timedelta blocks.

{strf:%FT%T}.000Z is the strf code for the ISO 8601 format, which is used when setting␣
↪the timestamp in an embed's JSON. Useful if you're creating embeds manually.
```

17.2 Timedelta

```
{td(OPTIONAL DATETIME OR UNIX):DATETIME}

{td:2020-01-01 00.00.00} as of 2019-11-25 would output 1 month, 5 days and 21 hours, or␣
↪the time until Midnight New Years Day

{td({m:trunc({unix}-3600)}):{strf:%Y-%m-%d %H.%M.%S}} -> 1 hour because we're just␣
↪subtracting 3600 seconds or 1 hour from the current time.
```

Timedelta blocks will output a readable time until said datetime will be reached.

CHAPTER

EIGHTEEN

BLANKS

Blanks are how we conditionally run blocks, this is due to tagscript running all blocks first, and not checking wether they should actually be run (Conditional Statements).

Danger: Probably the hardest concept you'll ever have to learn in tagscript, most people give up here.

If you think you can explain blanks feel free to add a dropdown with your explanation on it.

Asty' (With GIF)

What is that {=():} variable that you put in your code?

```
{=():}
{=(execution):{if({args}!=):c:echo Hello!}}
{{execution}}
```

In the simple example code above, if there is an argument, it will call the content of the execution variable, and because it has double brackets, it actually executes the command block.

If there is no argument, the else of the conditional if block that's in execution would return nothing, but when we call it with double brackets, it would output `{}` as plain text and we don't want that.

So to avoid that, we create a variable with no name and no content, that is being called when an else is blank.

Most common usage (please use this instead of the above!)

```
{=():}
{{if(boolean expression here):c:command here}}
Such as
{=():}
{{if(!={1}):c:echo Yay, you typed words: {args}}}
```

The above example runs the echo command block only if there was at least an argument (if the first argument is not empty, rather).

Reminder: `The {=():}` (blank variable) is there because otherwise when the condition evaluates to false, the line below it would return `{}` as plain text and would be outputted to the user, which we want to avoid.

_Leg3ndary

What Are Blanks

Tagscript is largely different from any other programming languages…

When you have any block that will send or alter its behaviour for example command blocks

```
eg {c:role add blah blah blah}
```

Normally you would do something like this

```
{if(condition):{c:role add blah blah blah}}
```

This will not work the c role will always add the role

It will run no matter what, no matter where you put it in the code, even if you put it in an `if` statement or in a `variable` it will still run. **TAGSCRIPT DOES NOT CARE**

So how do we stop this?

First we remove the brackets making it work

```
{if(condition):{c:role add blah blah blah}}
{if(condition):c:role add blah blah blah}
```

Notice now if we run it it will literally output `c:role add blah blah blah` but don't worry thats what we want!

Now all we have to do is actually allow carl to run it so we move it outside!

```
{if(condition):c:role add blah blah blah}
{{if(condition):c:role add blah blah blah}}
```

Now the command block will actually run because after outputting `c:role add blah blah blah` carl will add the brackets making it `{c:role add blah blah blah}`

Now all you have to do is add a `{=():}`, this will make it so when the condition fails instead of saying {} it will just say nothing.

Lets do one more example, I want to dm the command if no arguments are present:

Normally a person would do this:

```
{if(=={args}):{dm}}
```

But now knowing what we have to do lets fix this!

```
{if(=={args}):{dm}}
{if(=={args}):dm} first we remove the brackets!
{{if(=={args}):dm}} now we add them aroudn the outside
```

Finally we add a `{=():}` before it and we're done!

This can be used for many blocks like: `dm, redirect, require, blacklist, whitelist, del, silence, override` basically anything that will affect the overall block itself just replace `c:role add blah blah blah` with whatever block you want!`

Asty's Second Explanation

Deeper explanation about how to make any command/behavior block conditional using the Blank Variable `{=():}`

If you were to do `{if(this==that):{c:echo {args}}}`, then the command block would execute no matter what, because you put the entire block and it's a behavior block, and will ignore the fact it is inside of an if block.

Note: All action/behavior blocks like react(u), delete, silence, override, require/blacklist, dm, embed etc. will execute no matter where they are, if you are not using the blank workaround.

The way to make command blocks conditional is to have the if block return our command block's content as text, and then this text when returned would be evaluated as an actual block thanks to the `{` and `}` around the if block. When the condition evaluates to false, then the condition returns no text, so the entire block returns `{}`. This is where the blank variable `{=():}` comes in handy, as it would get called in that case and would return nothing. That's the workaround to allow you to execute a command block or any behavior block conditionally.

Essentially, we are doing `{if(this==that):c:echo {args}}`, which, when true is returned from the boolean equation (the check in the if block), returns the text c:echo {args}. And now for it to actually become a block it should execute when the condition is met, we wrap all of that between `{` and `}`. When the if block returns false, since no text would be returned, then with the extra brackets around it would become `{}`, which, if you have a blank variable above it `{=():}`, would actually call this variable that has no name and no content, effectively doing nothing.

Becomes:

```
{=():}
{{if(this==that):c:echo {args}}}
```

Replace the condition with yours and the command you're willing to execute in the snippet above. Don't copy-paste it without understanding what it does.

You only need 1 `{=():}` in your code!

CHAPTER

NINETEEN

SWITCHES

Switches are basically a way you can change a variable to something you've set before based on keywords.

Danger: If you have **anything** that looks relatively similar to this, you're doing it wrong. Keep reading for more info on why

```
{=(test_var):{if(keyword=={args}):something}}
{=(test_var):{if(keyword2=={args}):Tagscript|{test_var}}}
{=(test_var):{if(keyword3=={args}):Tagscript|{test_var}}}
{=(test_var):{if(keyword4=={args}):Tagscript|{test_var}}}
```

Raffael's Explanation (Original)

How to use variables to make a switch statement/membership test check

Let's say you want a tag that outputs a URL pretaining to one of the 16 Myers-Briggs Personality Type Indicators, depending on which personality type the user input. And if what the user inputted isn't one of those 16 types, show an error message. You could write a logical check for each one, like:

```
{if({1}==ENFJ):url|{if({1}==ISTP):url2|etc etc}}
```

but that can be messy to type out, and confusing to troubleshoot. Instead, let's use variables to check if the input matches one of them. First let's capitalize the first element of the input.

```
{=(u1):{upper:{1}}}
```

Next we'll assign our error message to a variable named the contents of the {u1} variable.

```
{=({u1}):Your input did not match one of the 16 Myers-Briggs Personality Types.}
```

Now we'll define our personality type variables. I'll set the base of the url to a variable to save space:

```
{=(url):https://www.verywellmind.com/}
{=(ENFJ):{url}enfj-extraverted-intuitive-feeling-judging-2795979}
{=(ENFP):{url}enfp-an-overview-of-the-champion-personality-type-2795980}
{=(ENTJ):{url}entj-personality-type-2795981}
{=(ENTP):{url}the-entp-personality-type-and-characteristics-2795982}
{=(ESFJ):{url}esfj-extraverted-sensing-feeling-judging-2795983}
{=(ESFP):{url}esfp-extraverted-sensing-feeling-perceiving-2795984}
{=(ESTJ):{url}estj-extraverted-sensing-thinking-judging-2795985}
{=(ESTP):{url}estp-extraverted-sensing-thinking-perceiving-2795986}
{=(INFJ):{url}infj-introverted-intuitive-feeling-judging-2795978}
```

(continues on next page)

(continued from previous page)

```
{=(INFP):{url}infp-a-profile-of-the-idealist-personality-type-2795987}
{=(INTJ):{url}intj-introverted-intuitive-thinking-judging-2795988}
{=(INTP):{url}intp-introverted-intuitive-thinking-perceiving-2795989}
{=(ISFJ):{url}isfj-introverted-sensing-feeling-judging-2795990}
{=(ISFP):{url}isfp-introverted-sensing-feeling-perceiving-2795991}
{=(ISTJ):{url}istj-introversion-sensing-thinking-judgment-2795992}
{=(ISTP):{url}istp-introverted-sensing-thinking-perceiving-2795993}
```

At the end, we'll call the contents of {u1} itself as a variable by putting brackets around it.

```
{{u1}}
```

This variable will either contain our error message since we previously assigned the error message to the variable with the name of {u1}, or that variable will have been overwritten by one of the 16 variables containing the URLs about the Myers-Briggs Personality types if {u1} was one of those types, and calling the variable will output the relevant link. No length checking or logical if statements needed.

Want to try it out?

Asty's Explanation

Switch Method Made Easy

Return a conditional response/content depending on the user's argument (words typed along with the custom command).

Prompt Example: We want to display a if the first argument is xyz, b when it's carl, or otherwise a default response.

Input:

`!mytag xyz`

Expected Output:

`a`

Here is how to do it without using a single if/conditional block, and only variables assignments and calls.

```
{=(l1):{lower:{1}}}

{=(r.{l1}):default response}
{=(r.xyz):a}
{=(r.carl):b}

{r.{l1}}
```

The first line simply sets the first argument (`{1}` is a short alias for `{args(1)}`) as lowercase, so it doesn't matter if the user typed `XYZ` or `xYz` or `xyz`, it will still consider it as `xyz`.

Then, I created variables with `r.` as their name prefix just so I'm sure the rest of the name won't conflict with blocks that already exist in TagScript.

The principle of the switch method (which is explained thoroughly but in a more complex way in the pinned messages in this channel) is to basically have a variable that returns content based on the user arguments.

`r.{l1}` here creates a default response for when the argument specified doesn't meet any other below it. For example, if the user types `xyz` as the first argument, the `r.{l1}` variable will be overwritten with the content of `r.xyz`, effectively returning a in our case instead of the default response, because it exists.

In our case, when neither `xyz` or `carl` was specified as the first argument, then it will send the default response.

And then the last line is what actually calls the response to display it to the user.

You could store it in a variable you'd name something like `response` or `output`, if you wanted to call it somewhere else, in your embed's description field for example.

CHAPTER

TWENTY

USERNAME#DISCRIMINATOR

MAKE SURE TO ADD THIS TO THE index.rst FILE, LOOK FOR toctree AND CustomTags/TEMPLATE

Table of Tags

20.1 TAG NAME

ADD A DESCRIPTION

Source Code

```
PUT SOURCE CODE HERE
```

Tag Import

CHAPTER

TWENTYONE

RAFFAEL#1372

Table of Tags

21.1 Original AFK COMMAND

Made an "afk" tag. When it's set to on it will inform anyone that mentions you that you are afk. Setting it to `off` will turn it off. If you type a message after on it will be set as your away message, but if you leave it blank the tag will use the default away message.

Source Code

```
{=(L1):{lower:{1}}}
{=(A2+):{replace(|, - ):{args(2+)}}}
{=(error):You must follow the `afk` command with either `on` or `off`.}
{=(on):add}
{=(off):del}
{=(template):c:ar {{L1}} {user(id)}>{if({L1}==on): {if({list(0):{join(,):{A2+}}}!=):{A2+}
↪|{replace(|, - ):{user}} is afk right now, send them a PM or wait for them to return.}}
↪}
{=(sel):{if({contains({L1}):on off}==false):error|{template}}}
{override}{{sel}}
```

Tag Import

21.2 Back 4 Blood Card Lookup

A Back 4 Blood card lookup command. Will display the card that best matches the provided search term alphabetically. Displays an error message if the provided search term does not match a card's name or a substring of that name.

For when you can't recall the specifics of a card, or want to discuss it within a Discord server without having to type out the card's description or stats.

I will attempt to keep the card images updated as future patches occur. These card images are current as of 2021-10-16.

This custom command is not affiliated with or endorsed by Back 4 Blood, Warner Bros. and it's divisions, or Turtle Rock Studios. I do not own these images, I am using them under the constraints of Fair Use as established in the Copyright Act of 1976, for nonprofit educational purposes.

The image below shows the command invocation for example purposes. This invocation will be deleted upon execution in the linked version of this tag.

Source Code

```
{delete}
{=(comment):This custom command is not affiliated with or endorsed by Back 4 Blood,␣
↪Warner Bros. and it's divisions, or Turtle Rock Studios. I do not own these images, I␣
↪am using them under the constraints of Fair Use as established in the Copyright Act of␣
↪1976, for nonprofit educational purposes.}
{=(cardnames):Cadmin_reload Cadrenaline_fueled Cammo_belt Bammo_drop Cammo_for_all Cammo_
↪mule Cammo_pouch Cammo_scavenger Cammo_stash Camped_up Cantibiotic_ointment Cavenge_
↪the_fallen Cbatter_up Cbattle_lust Cbelt_clip Cberserker Cbody_armor Cbodyguard Cbomb_
↪squad Cbounty_hunter Cbox_o'_bags Cbravado Cbrazen Cbreakout Cbroadside Cbuckshot_
↪bruiser Ccanned_goods Ccharitable_soul Cchemical_courage Ccocky Ccold_brew_coffee␣
↪Ccombat_knife Ccombat_medic Ccombat_training Ccompound_interest Cconfident_killer␣
↪Ccontrolled_movement Ccopper_scavenger Ccross_trainers Cdash Cdefensive_maneuver␣
↪Cdemolitions_expert Pdoc Cdouble_grenade_pouch Cdown_in_front! Cdurable Bdusty's_
↪customs_assault_rifle Bdusty's_customs_handgun Bdusty's_customs_lmg Bdusty's_customs_
↪shotgun Bdusty's_customs_smg Bdusty's_customs_sniper_rifle Cemt_bag Cenergy_bar␣
↪Cenergy_drink Pevangelo Cexperienced_emt Bextra_padding Cface_your_fears Cfanny_pack␣
↪Cfield_surgeon Cfire_in_the_hole! Cfit_as_a_fiddle Cfleet_of_foot Cfresh_bandage␣
↪Cfront_sight_focus Cglass_cannon Cgrenade_pouch Cgrenade_training Cgroup_therapy Cguns_
↪out Chazard_pay Bhazard_suit Cheadband_magnifier Cheavy_attack Cheavy_hitter Bhell_can_
↪wait Chellfire Pheng Chi_vis_sights Chighwayman Bhired_gun Phoffman Pholly Chunker_
↪down Chydration_pack Chyper-focused Cignore_the_pain Cimprovised_explosives Cin_the_
↪zone Cinspiring_sacrifice Pjim Pkarlee Ckiller's_instinct Cknowledge_is_power Clarge_
↪caliber_rounds Clife_insurance Cline_'em_up Clucky_pennies Cmad_dash Cmag_carrier Cmag_
↪coupler Cmagician's_apprentice Cmandatory_pt Cmarathon_runner Cmarked_for_death Cmean_
↪drunk Cmeatgrinder Cmedical_expert Cmedical_professional Cmeth_head Cmiraculous_
↪recovery Pmom Cmoney_grubbers Cmotorcycle_helmet Cmotorcycle_jacket Cmugger Cmultitool␣
↪Cnatural_sprinter Cneeds_of_the_many Cnumb Coffensive_scavenger Colympic_sprinter Con_
↪your_mark Coptics_enthusiast Cover-protective Coverwatch Cpadded_suit Cpatient_hunter␣
↪Cpep_in_your_step Cpep_talk Cpinata Cpoultice Cpower_reload Cpower_strike Cpower_swap␣
↪Cpumped_up Cpyro Cquick_kill Creckless Creckless_strategy Creload_drills Crhythmic_
↪breathing Cridden_slayer Crolling_thunder Crousing_speech Crun_and_gun Crun_like_hell␣
↪Csadist Csadistic Csaferoom_recovery Cscar_tissue Cscattergun_skills Cscrewdriver␣
↪Csecond_chance Cshare_the_wealth Psharice Cshell_carrier Cshooting_gloves Cshoulder_
↪bag Cshredder Csilver_bullets Bslippery_when_wet Cslugger Csmelling_salts Cspeed_demon␣
↪Cspiky_bits Csteady_aim Cstealthy_passage Cstimulants Cstock_pouch Csunder Csuperior_
↪cardio Csupport_scavenger Csurplus_pouches Ctactical_vest Ctool_belts Ctrigger_control␣
↪Ctrue_grit Ctunnel_vision Ctwo_is_one_and_one_is_none Burgent_care Cutility_belt␣
↪Cutility_scavenger Cvanguard Cvitamins Pwalker Cweapon_scavenger Cweaponsmith Cwell_
↪fed Cwell_rested Cwidemouth_magwell Bwindfall Cwooden_armor Cwounded_animal }
```

(continues on next page)

(continued from previous page)

```
{=(searchterm):{replace( ,_):{lower:{args}}}}
{if({in({searchterm}):{cardnames}}==true):https://raw.githubusercontent.com/Raffael7777/
↪B4B-Card-Images/main/{replace(P,Cleaner%20Cards/):{replace(B,Burn%20Cards/):{replace(C,
↪Campaign%20Cards/):{cardnames({index({searchterm}):{replace({searchterm},. {searchterm}
↪ .):{cardnames}}})}}}}.png|`{replace(_, ):{searchterm}}` does not appear to match a␣
↪card name or part of a card name. Double check your input and spelling and try again.}
```

Tag Import

CHAPTER

TWENTYTWO

_LEG3NDARY#5759

Table of Tags

22.1 Shell Cloud Command

Working this out took a lot of time, however I'm proud to show the tagscript community the first tag that can save data through the bot, **you don't have to do any setting up.** (Other then importing of course)

You might recall **Raffael** made a tag where you would setup a role and then guess the number, you also might remember a user named **Squidtoon** made an api that did similar things however he removed his project from repl where it was hosted…

What?

You can use this tag to do something similar to the cbsays tag where you can post a shell with text on it, you can also view other users posts or the latest post!

Why? This is just the general idea of using an api and python to interact through embed images, for those of you who don't know what an API is, its an "application program interface", you send a request to a site which then can give you something back, thats basically what I did. You send a request to a repl, the program will then save it to another further external cloud db known as mongodb (There are problems with saving data through repl which is why this part is needed).

Usage:

```
?shell add <Word>
Your word must be less then or 10 characters
?shell show <Index>
View already posted words, must be a number eg. view the 7th word added with the add␣
↪command
USE 0 OR LATEST TO VIEW LATEST POST
?shell cache
Shows the Cache which you can use to view words with the show command
```

Warning.

People are naturally stupid. In short people will be able to use words that shouldn't be used. For this reason I highly recommend this tag be marked nsfw.

Tag does not use any commands.

Source Code

~~~
{=(Comment): Tag by _Leg3ndary :D Nothing should be edited in this tag... other then the variables below...}

{embed(color):{user(color)}}
{=(prefix):?}
{=(command_name):shell}

{=(Comment): Don't touch anything below}

{=():}

{=(len_args):{index():{args} }}
{=(type):{lower:{args(1)}}}

{=({type}):/error}
{=(dev/show):dev/show}
{=(cache):dev/cache}
{=(show):show/{if(latest=={lower:{2}}):0|{args(2)}}/{unix}}
{=(add):add/{if(=={2}):Blank|{args(2)}}/{unix}}

{{if(/error=={{type}}):embed(title):Error}}
{{if(/error=={{type}}):embed(description):Subcommand {if(=={args(1)}):__`Blank`__|`{lower:{args(1)}}`} not found...
Available options are:
```asciidoc
= {prefix}{command_name} add <Word> =
[ Your word must be less then or 10 characters ]
= {prefix}{command_name} show <Index> =
[ View already posted words, must be a number eg. view the 7th word added with the add command ]
[ USE 0 OR LATEST TO VIEW LATEST POST THIS MIGHT NOT WORK THOUGH ]
= {prefix}{command_name} cache =
[ Shows the Cache which you can use to view words with the show command ]
```}}

{{if(dev/cache=={{type}}):embed(title):Cache}}
{{if(dev/cache=={{type}}):embed(description):This link shows the cache which will be formatted like so:
~~~
{"1":"Carl-bot","2":"Hello"}
```
[__`Cache`__](https://leg3ndarytagscript.tenshibot.repl.co/dev/cache)}}

{=(endpoint):{if(/error!={{type}}):{{type}}}}

{=(api_endpoint):https://leg3ndarytagscript.tenshibot.repl.co/{endpoint}}
```

Tag Import

22.2 Suspicion Command

Check for alts using a system of checks with each being given their own weight, in addition see other info about the user to manually determine if a user is “sus”.

```
{=(ud.seconds.points):3} {=(README):Users account age is under 24 hours old, usually an␣
↪alt MAKE THIS THE BIGGEST NUMBER}
{=(ud.minutes.points):2} {=(README):Users account age is 1-30 days old, could be an alt␣
↪would recommend keeping an eye on the user}
{=(ud.hours.points):1} {=(README):Users account age is 1-12 months old, again could be␣
↪an alt but not likely}
{=(ud.days.points):0} {=(README):Users account age is 1+ years old, most likely not an␣
↪alt but could be one}
{=(ujd.seconds.points):2} {=(README):Users account age in server is under 24 hours old,␣
↪new memberMAKE THIS THE BIGGEST NUMBER}
{=(ujd.minutes.points):1} {=(README):Users account age in server is 1-30 days old, not a␣
↪new member but not really an old member}
{=(ujd.hours.points):0} {=(README):Users account age in server is 1-12 months old, would␣
↪be considered seasoned or at least an old member}
{=(ujd.days.points):0} {=(README):Users account age in the server is 1+ years old, old␣
↪member}
{=(ua.default.points):3} {=(README):Is the users avatar a default one? Most alts don't␣
↪change pfps which is a dead giveaway}
{=(ua.nitro.points):-1} {=(README):Does the user have a .gif avatar? Not many people␣
↪have nitro for an alt(s) Negative number here because user has nitro}
{=(un.points):1} {=(README): Has the user changed his/her nickname? Alts don't usually␣
↪change their nickname, Disable this with "0" if needed}
```

Source Code

```
{=(COMMENT): Change the prefix and the recommended action taken if needed}
{=(tag.prefix):?}
{=(preset.punishment):warn}
{=(preset.reason):Please don't use alts as they are against our rules.}


{=(COMMENT): None of these should be changed unless you know what your doing and even␣
↪then not really needed basically finds the basic vars you need and sets a var with all␣
↪the default variable avatars}
{=(user.days):{td:{target(created_at)}}}
{=(user.join.days):{td:{target(joined_at)}}}
{=(default.avatars):https://cdn.discordapp.com/embed/avatars/0.png https://cdn.
↪discordapp.com/embed/avatars/1.png https://cdn.discordapp.com/embed/avatars/2.png␣
↪https://cdn.discordapp.com/embed/avatars/3.png https://cdn.discordapp.com/embed/
↪avatars/4.png}

{=(Comment): This is a limiter to prevent the embed from breaking change it to increase␣
↪the number of roles seen-or decrease it}
{=(limiter):15}
{=(user.roleids.sub):{target(roleids)}}
{=(user.roleids.stuff):{index(abc):{target(roleids)} abc}}
{=(user.roleids.stuff):{if({user.roleids.stuff}>{limiter}):{replace({user.roleids.sub(+
↪{m:trunc({user.roleids.stuff}-{limiter})})},):{target(roleids)}|{target(roleids)}}}}
```

(continues on next page)

(continued from previous page)

```
{=(user.roleids.ping):<@&{replace( ,> <@&):{user.roleids.stuff}>}}
{=(user.roleids.ping):{if({user.roleids.ping}==<@&>):None|{replace(<@&>,):{user.roleids.
↪ping}}}}
{=(user.roleids.list):{target(roleids)}}

{=(COMMENT):Sets the sus.score to 0 so that it can calculate the later values in this␣
↪command}
{=(sus.score):0}

{=(COMMENT): THIS IS THE POINTS SYSTEM, IT IS ESSENTIAL YOU READ AND UNDERSTAND THIS}
{=(COMMENT): This determines what level suspicion the user has, the bigger the number␣
↪the more suspicion 0 means nothing and it won't affect the final variable itself, use␣
↪that to disable parts of the command, you can have values with negatives to make the␣
↪suspicion score smaller (will make the final score less sus) more is explained about␣
↪it beside the var itself}
{=(ud.seconds.points):3} {=(README):Users account age is under 24 hours old, usually an␣
↪alt MAKE THIS THE BIGGEST NUMBER}
{=(ud.minutes.points):2} {=(README):Users account age is 1-30 days old, could be an alt␣
↪would recommend keeping an eye on the user}
{=(ud.hours.points):1} {=(README):Users account age is 1-12 months old, again could be␣
↪an alt but not likely}
{=(ud.days.points):0} {=(README):Users account age is 1+ years old, most likely not an␣
↪alt but could be one}
{=(ujd.seconds.points):2} {=(README):Users account age in server is under 24 hours old,␣
↪new memberMAKE THIS THE BIGGEST NUMBER}
{=(ujd.minutes.points):1} {=(README):Users account age in server is 1-30 days old, not a␣
↪new member but not really an old member}
{=(ujd.hours.points):0} {=(README):Users account age in server is 1-12 months old, would␣
↪be considered seasoned or at least an old member}
{=(ujd.days.points):0} {=(README):Users account age in the server is 1+ years old, old␣
↪member}
{=(ua.default.points):3} {=(README):Is the users avatar a default one? Most alts don't␣
↪change pfps which is a dead giveaway}
{=(ua.nitro.points):-1} {=(README):Does the user have a .gif avatar? Not many people␣
↪have nitro for an alt(s) Negative number here because user has nitro}
{=(un.points):1} {=(README): Has the user changed his/her nickname? Alts don't usually␣
↪change their nickname, Disable this with "0" if needed}

{=(COMMENT):Checking and determining how old the account is and then adding sus points␣
↪based on it}
{=(sus.score):{if({in(seconds ago):{user.days}}==true):{m:{sus.score}+{ud.seconds.points}
↪}|{sus.score}}}
{=(sus.score):{if({in(minutes ago):{user.days}}==true):{m:{sus.score}+{ud.minutes.points}
↪}|{sus.score}}}
{=(sus.score):{if({in(hours ago):{user.days}}==true):{m:{sus.score}+{ud.hours.points}}|
↪{sus.score}}}
{=(sus.score):{if({in(days ago):{user.days}}==true):{m:{sus.score}+{ud.days.points}}|
↪{sus.score}}}

{=(COMMENT):Same as above except its checking account age in the server or how long he/
↪she has been in the server}
{=(sus.score):{if({in(seconds ago):{user.join.days}}==true):{m:{sus.score}+{ujd.seconds.
↪points}}|{sus.score}}}
```

(continues on next page)

(continued from previous page)

```
{=(sus.score):{if({in(minutes ago):{user.join.days}}==true):{m:{sus.score}+{ujd.minutes.
↪points}}|{sus.score}}}
{=(sus.score):{if({in(hours ago):{user.join.days}}==true):{m:{sus.score}+{ujd.hours.
↪points}}|{sus.score}}}
{=(sus.score):{if({in(days ago):{user.join.days}}==true):{m:{sus.score}+{ujd.days.points}
↪}|{sus.score}}}

{=(COMMENT): Checking if the user has a nitro pfp or if its just a regular default pfp␣
↪in addition we'll check the discriminator as if it has a 1111 or 0001 or something␣
↪like that we know they probably changed it and has nitro}
{=(sus.score):{if({in({target(avatar)}):{default.avatars}}==true):{m:{sus.score}+{ua.
↪default.points}}|{sus.score}}}
{=(sus.score):{if({in(.gif):{target(avatar)}}==true):{m:{sus.score}+{ua.nitro.points}}|
↪{sus.score}}}
{=(COMMENT): This list below is what we'll be checking... Edit as you please it will␣
↪also check if we've determined he/she already has nitro so we don't double it up}
{=(sus.discrim):0001 0002 0003 0004 0005 0006 0007 0008 0009 1111 2222 3333 4444 5555␣
↪6666 7777 8888 9999 2020 2021 1000 2000 3000 4000 5000 6000 7000 8000 9000}
{=(sus.score):{and({contains({replace({user(name)}#,):{user(proper)}}):{sus.discrim}}
↪==true|{in(.gif):{target(avatar)}}==false):{m:{sus.score}+{ua.nitro.points}}|{sus.
↪score}}}

{=(COMMENT):Has the user changed his or her name since joining? Again you can disable␣
↪this if you want by changing un.points to 0}
{=(sus.score):{if({target}=={target(name)}):{m:{sus.score}+{un.points}}|{sus.score}}}

{=(COMMENT):Checking how many roles the user has, had a problem if the user had 1 or 0␣
↪roles would output 0 no matter what so the bottom block checks if its 1 or 0 and␣
↪changes the above value to the correct one}
{=(user.roleids.number):{index($$$):{{target(roleids)}} $$$}}
{=(user.roleids.number):{if({user.roleids.list(1)}=={user.roleids.list(2)}):{if({user.
↪roleids.list(1)}==):0|1}|{user.roleids.number}}}

{=(COMMENT):Taking all the scores checking if there negative and then adding if they aren
↪'t This determines the final percentage and embed color which is why you must follow␣
↪the points system correctly ^ find above}
{=(total.score):{m:{if({m:sgn({ud.seconds.points})}==-1):0|{ud.seconds.points}}+{if(
↪{m:sgn({ujd.seconds.points})}==-1):0|{ujd.seconds.points}}+{if({m:sgn({ua.default.
↪points})}==-1):0|{ua.default.points}}+{if({m:sgn({ua.nitro.points})}==-1):0|{ua.nitro.
↪points}}+{if({m:sgn({un.points})}==-1):0|{un.points}}}}

{=(COMMENT):Finally determining the percentage since truncate can't cut of to a certain␣
↪decimal it multiplies by 10000 then truncates and divides by 100 which gives it the␣
↪decimal, this could be in one block but I've left it to multiple so you can edit/
↪better understand it}
{=(sus.score):{m:{sus.score}/{total.score}}}
{=(sus.score):{if({target(proper)}=={server(owner)}):0.00|{sus.score}}} {=(README): Just␣
↪checking if the person is the owner}
{=(sus.score):{m:{sus.score}*10000}}
{=(sus.score):{m:trunc({sus.score})}}
{=(sus.score):{m:{sus.score}/100}}
```

(continues on next page)

(continued from previous page)

~~~
{=(COMMENT):Embed color, don't touch if you don't know how it works, if you want to have␣
↪just one color change the bottom block with the hex you want}
{=(embed.color):{if({sus.score}<=20.001):7ED321|{if({sus.score}<=40.001):BBDD1F|{if({sus.
↪score}<=60.001):F8E71C|{if({sus.score}<=80.001):E4751C|D0021B}}}}}
{embed(color):#{embed.color}}

{=(COMMENT):The final punishment if recommended}
{=(preset.punishment.final):{if({sus.score}>=90.001):Most likely an alt, command to
↪{preset.punishment}: ```
{tag.prefix}{preset.punishment} {target(id)} {preset.reason}
```|}}
~~~

Tag Import

CHAPTER

TWENTYTHREE

ASTY'#8926

Table of Tags

23.1 The Best Embed Maker Command

Have you ever dreamt of creating an Embed Message with images, a footer, a custom side color… easily with just a command?

As you may already know, Carl-bot has the embed command that limits you to only a title, a description and a side color, and the `cembed` command that requires you to know about what `JSON` even is (which is not the majority, let's face it), and is a pain to use on a regular basis (or even once to be honest).

This is why out of frustration and as a huge QoL (Quality of Life) improvement for you all, I built this `buildembed` custom command (`be` for short), which is very easy and convenient to use, while being powerful!

Features

- Each field name has Substring Matching, meaning you can for example type t for title, desc for description, c or col for color and it will still recognize it!
- You can specify any field in the order of your choice! Unlike other embed commands, if you want to specify a footer, then a thumbnail, then a title, you can!
- For the color field, you can specify either a color name, a hexadecimal code (like #eeaaee), or even self for your own color.
- Custom-made blocks to use in some fields, such as user {avatar} and server {icon} in image fields, {server} and {channel} in text fields, and {channel(mention)} in the Description field.
- Even if you don't have Nitro, ability to use custom emojis from other servers, by using <emoji (literally) followed by the emoji ID and >.

Usage

```
!be fieldName:Content|fieldName:Content|fieldName:Content...
```

Available Fields and Colors

Source Code

```
{=(COMMENT):User Settings}
{=(prefix):!}
{=(tagName):buildembed|be}

{=(offsetFromUTC):0}

{=(example):{prefix}{tagName} title:My title|desc:My description|thumb:https://i.imgur.
↪com/0l0ZBCm.png|footer:My footer|color:blurple}

{=(defaultColor):2f3136}
{=(defaultFooter):{prefix}{tagName} title:My title|desc:My description|color:red}
{=(defaultFooter):{example}}

{=(COMMENT):Only the roles specified in there (separated by a comma) will be able to use␣
↪that command. Leave empty if you don't mind.}
{=(requiredRoles):}
{=(COMMENT):The error message to output to the user when they don't have the required␣
↪role. Leaving it empty will react to the user's message with the  emoji, and leaving a␣
↪space " " character won't return an error message at all}
{=(requiredRolesErrorMessage):}


{=(COMMENT):Adding or subtracting (depending on if the offset is positive or negative)␣
↪the offset from UTC specified above to the current unix time.}
{=(unixTimezone):{m:trunc({unix}+({offsetFromUTC}*3600))}}

{=(defaultTime):{strf({unixTimezone}):%FT%T.000Z}}

{=(delimiter):|}

{=(r):replace}
{=(i):index}
{=(l):lower}
{=(j):join}
{=(ct):contains}
{=(fb):None}
{=(del):{delimiter}}
{=(pl):%&$}
{=(uav):{user(avatar)}}
{=(sic):{server(icon)}}

{=(COMMENT):Curly brace snippet}
{=( ):{ }}
{=(b):{ (1)}}
{=(d):{ (2)}}

{=(emojiPrefix):<emoji}

{=(isAChannelSpecified):{in(true true):{in(<#):{1}} {in(>):{1}}}}
{=(channelToSend):{{r}(false,{channel(id)}):{{r}(true,{1}):{isAChannelSpecified}}}}


{=(COMMENT):Separating arguments into their own list variables. Replaces the delimiter␣
↪with ~ in the process.}
```

(continues on next page)

(continued from previous page)

```
{=(sanitizedArgs):{{r}({emojiPrefix},<:emoji:):{{r}(",\"):{args}}}}
{=(argumentToParse):{{if({isAChannelSpecified}==true):sanitizedArgs(2+)|sanitizedArgs}}}
{=(jargs):{{r}({del},~):!~{argumentToParse}}}
{=(a1):{list(1):{jargs}}}
{=(a2):{list(2):{jargs}}}
{=(a3):{list(3):{jargs}}}
{=(a4):{list(4):{jargs}}}
{=(a5):{list(5):{jargs}}}
{=(a6):{list(6):{jargs}}}
{=(a7):{list(7):{jargs}}}
{=(a8):{list(8):{jargs}}}
{=(a9):{list(9):{jargs}}}
{=(a10):{list(10):{jargs}}}
{=(a11):{list(11):{jargs}}}
{=(a12):{list(12):{jargs}}}

{=(fieldsName):image title titleurl name color nameicon nameurl description thumbnail␣
↪footer footericon timestamp {fb} !}

{=(COMMENT):Processes a substring matching on the keyword/field name of every argument␣
↪up to the 12th and returns it.}
{=(COMMENT):Raw argument word, substring matched word, and then word content}
{=(w1):{{l}:{a1(1)::}}}
{=(word1):{fieldsName({{i}({w1}):{{r}({w1},. {w1} .):{fieldsName}}})}}
{=(content1):{a1(2+)::}}

{=(w2):{{l}:{a2(1)::}}}
{=(word2):{fieldsName({{i}({w2}):{{r}({w2},. {w2} .):{fieldsName}}})}}
{=(content2):{a2(2+)::}}

{=(w3):{{l}:{a3(1)::}}}
{=(word3):{fieldsName({{i}({w3}):{{r}({w3},. {w3} .):{fieldsName}}})}}
{=(content3):{a3(2+)::}}

{=(w4):{{l}:{a4(1)::}}}
{=(word4):{fieldsName({{i}({w4}):{{r}({w4},. {w4} .):{fieldsName}}})}}
{=(content4):{a4(2+)::}}

{=(w5):{{l}:{a5(1)::}}}
{=(word5):{fieldsName({{i}({w5}):{{r}({w5},. {w5} .):{fieldsName}}})}}
{=(content5):{a5(2+)::}}

{=(w6):{{l}:{a6(1)::}}}
{=(word6):{fieldsName({{i}({w6}):{{r}({w6},. {w6} .):{fieldsName}}})}}
{=(content6):{a6(2+)::}}

{=(w7):{{l}:{a7(1)::}}}
{=(word7):{fieldsName({{i}({w7}):{{r}({w7},. {w7} .):{fieldsName}}})}}
{=(content7):{a7(2+)::}}

{=(w8):{{l}:{a8(1)::}}}
{=(word8):{fieldsName({{i}({w8}):{{r}({w8},. {w8} .):{fieldsName}}})}}
```

(continues on next page)

(continued from previous page)

```
{=(content8):{a8(2+)::}}

{=(w9):{{l}:{a9(1)::}}}
{=(word9):{fieldsName({{i}({w9}):{{r}({w9},. {w9} .):{fieldsName}}})}}
{=(content9):{a9(2+)::}}

{=(w10):{{l}:{a10(1)::}}}
{=(word10):{fieldsName({{i}({w10}):{{r}({w10},. {w10} .):{fieldsName}}})}}
{=(content10):{a10(2+)::}}

{=(w11):{{l}:{a11(1)::}}}
{=(word11):{fieldsName({{i}({w11}):{{r}({w11},. {w11} .):{fieldsName}}})}}
{=(content11):{a11(2+)::}}

{=(w12):{{l}:{a12(1)::}}}
{=(word12):{fieldsName({{i}({w12}):{{r}({w12},. {w12} .):{fieldsName}}})}}
{=(content12):{a12(2+)::}}

{=(COMMENT):Fields variables from input. This is where we call our default values as␣
↪their content if any.}
{=(COMMENT):Order of the variables assignments in the same order as the embed builder.}

{=(f.name):}
{=(f.nameicon):}
{=(f.nameurl):}

{=(f.title):}
{=(f.titleurl):}

{=(f.description):}

{=(f.image):}
{=(f.thumbnail):}

{=(f.footer):}
{=(f.footericon):}

{=(f.timestamp):}

{=(f.color):{defaultColor}}

{=(f.{word1}):{content1}}
{=(f.{word2}):{content2}}
{=(f.{word3}):{content3}}
{=(f.{word4}):{content4}}
{=(f.{word5}):{content5}}
{=(f.{word6}):{content6}}
{=(f.{word7}):{content7}}
{=(f.{word8}):{content8}}
{=(f.{word9}):{content9}}
{=(f.{word10}):{content10}}
{=(f.{word11}):{content11}}
```

(continues on next page)

(continued from previous page)

```
{=(f.{word12}):{content12}}

{=(COMMENT):Allowing the word "now" to be used for the "timestamp" field.}
{=(f.timestamp):{if({{l}:{f.timestamp}}==now):{{r}(now,{defaultTime}):{{l}:{f.timestamp}}
↪}}}

{=(COMMENT):Although I'm allowing "self" and other keywords for user color, this is␣
↪allowing the custom user color TagScript block to work beforehand.}
{=(f.color):{{r}({b}user(color){d},{user(color)}):{{j}():{f.color}}}}

{=(COMMENT):Allowing custom blocks like user avatar and server icon to be used in image␣
↪fields as arguments.}
{=(f.image):{{r}({b}icon{d},{sic}):{{r}({b}server(icon){d},{b}icon{d}):{{r}({b}avatar{d},
↪{uav}):{{r}({b}user(avatar){d},{b}avatar{d}):{f.image}}}}}}

{=(f.nameicon):{{r}({b}icon{d},{sic}):{{r}({b}server(icon){d},{b}icon{d}):{{r}({b}avatar
↪{d},{uav}):{{r}({b}user(avatar){d},{b}avatar{d}):{f.nameicon}}}}}}
{=(f.nicon):{{r}({b}icon{d},{sic}):{{r}({b}server(icon){d},{b}icon{d}):{{r}({b}avatar{d},
↪{uav}):{{r}({b}user(avatar){d},{b}avatar{d}):{f.nico}}}}}}

{=(f.thumbnail):{{r}({b}icon{d},{sic}):{{r}({b}server(icon){d},{b}icon{d}):{{r}({b}avatar
↪{d},{uav}):{{r}({b}user(avatar){d},{b}avatar{d}):{f.thumbnail}}}}}}

{=(f.footericon):{{r}({b}icon{d},{sic}):{{r}({b}server(icon){d},{b}icon{d}):{{r}({b}
↪avatar{d},{uav}):{{r}({b}user(avatar){d},{b}avatar{d}):{f.footericon}}}}}}

{=(COMMENT):Allowing custom blocks in text fields.}
{=(f.title):{{r}({b}channel{d},{channel}):{{r}({b}server{d},{server}):{{r}({b}user{d},
↪{user}):{f.title}}}}}
{=(f.name):{{r}({b}channel{d},{channel}):{{r}({b}server{d},{server}):{{r}({b}user{d},
↪{user}):{f.name}}}}}
{=(f.footer):{{r}({b}channel{d},{channel}):{{r}({b}server{d},{server}):{{r}({b}user{d},
↪{user}):{f.footer}}}}}
{=(f.description):{{r}({b}channel(mention){d},{channel(mention)}):{{r}({b}channel{d},
↪{channel}):{{r}({b}server{d},{server}):{{r}({b}mention{d},{user(mention)}):{{r}({b}
↪user(mention){d},{b}mention{d}):{{r}({b}user{d},{user}):{f.description}}}}}}}}

{=(COMMENT):Spaces out the color, then removes the first and last added space}
{=(sepColor):{{r}(, ):{f.color}}}
{=(sepColor):{sepColor(2+)}}
{=(sepColor):{sepColor(+-1)}}
{=(COMMENT):Sanitize our color to only keep our first 6 characters of it, remove the #␣
↪sign if any and make every letter uppercase}
{=(sanitizedColor):{{r}(#,):{lower:{{j}():{sepColor}}}}}

{=(COMMENT):Dictionary of color names, returned as hex. https://htmlcolorcodes.com/color-
↪names/}
{=(cl.{f.color}):{sanitizedColor(+6)}}
{=(cl.white):FFFFFF}
{=(cl.silver):C0C0C0}
{=(cl.gray):808080}
{=(cl.grey):{cl.gray}}
```

(continues on next page)

(continued from previous page)

```
{=(cl.slategray):708090}
{=(cl.slategrey):{cl.slategray}}
{=(cl.black):000000}
{=(cl.red):FF0000}
{=(cl.yellow):FFFF00}
{=(cl.lime):00FF00}
{=(cl.green):008000}
{=(cl.cyan):00FFFF}
{=(cl.blue):0000FF}
{=(cl.navy):000080}
{=(cl.purple):800080}
{=(cl.salmon):FA8072}
{=(cl.pink):FFC0CB}
{=(cl.coral):FF7F50}
{=(cl.orange):FFA500}
{=(cl.gold):FFD700}
{=(cl.magenta):FF00FF}
{=(cl.violet):EE82EE}
{=(cl.indigo):4B0082}
{=(cl.slateblue):6A5ACD}
{=(cl.midnightblue):191970}
{=(cl.wheat):F5DEB3}
{=(cl.chocolate):F5DEB3}

{=(cl.discord):2F3136}
{=(cl.embed):{cl.discord}}
{=(cl.blurple):7289DA}

{=(cl.self):{user(color)}}
{=(cl.myself):{cl.self}}
{=(cl.me):{cl.self}}

{=(finalHexColor):{upper:{cl.{lower:{{j}():{f.color}}}}}}

{=(COMMENT):Spacing out our hexadecimal code}
{=(spacedHex):{{r}(F,15):{{r}(E,14):{{r}(D,13):{{r}(C,12):{{r}(B,11):{{r}(A,10):{{r}(, ):
↪{finalHexColor}}}}}}}}
{=(spacedHex):{spacedHex(2+)}}
{=(spacedHex):{{j}(~):{spacedHex(+-1)}}}

{=(COMMENT):Hex to decimal part}
{=(finalDecimalColor):{m:trunc({{j}():{spacedHex(0):~}*16^0 + {spacedHex(-1):~}*16^1 +
↪{spacedHex(-2):~}*16^2 + {spacedHex(-3):~}*16^3 + {spacedHex(-4):~}*16^4 + {spacedHex(-
↪5):~}*16^5)}}}

{=(finalJSON):{ "url":"{f.titleurl}",
    "title": "{f.title}",
    "description": "{f.description}",
    "thumbnail": {
        "url": "{f.thumbnail}"
        },
    "image": {
```

(continues on next page)

(continued from previous page)

```
        "url": "{f.image}"
        },
    "author": {
        "name":"{f.name}",
        "url": "{f.nameurl}",
        "icon_url": "{f.nameicon}"
        },
    "color": {finalDecimalColor},
    "footer": {
        "icon_url": "{f.footericon}",
        "text": "{f.footer}"
        },
    "timestamp": "{f.timestamp}"
        }}

{=(allFields):{{j}():{{r}({fb},):{word1} {word2} {word3} {word4} {word5} {word6} {word7}
↪{word8} {word9} {word10} {word11} {word12}}}}

{=(areAllFieldsEmpty):{{ct}({pl}):{allFields}{pl}}}
{=(isNoArg):{{ct}({pl}):{a1}{pl}}}

{=(errJSON):{ "description": "No field detected.\nExample:
    `{example}`",
    "color": {finalDecimalColor}
}
}

{=(errJSON):{ "fields": [
    { "name":
        "Features","value":"• **Substring Matching** on each field name (`t` works for␣
↪`title`, `desc` for `description`, etc.)\n• Fields names can all be filled and in␣
↪**any order** of your choice!\n• Built-in **error messages**, to let you know what's␣
↪wrong and when.\n• Ability to specify a color name, hexadecimal color, or even `self`␣
↪for your own color.\n• **Custom-made blocks** to use in some fields, like user `
↪{avatar}` and server `{icon}` in image fields, `{server}` and `{channel}` in text␣
↪fields, and `{channel(mention)}` in the Description field.\n• Even for non-Nitro users,
↪ ability to use **custom emojis** **from other servers**, by using `<emoji` followed␣
↪by the **emoji ID** and `>`. For example, `<emoji803680930841362442>` would display
↪<:TagScript:803680930841362442>.\n","inline":false
    },
    { "name":
            "Usage",
        "value":
            "`!be fieldName:Content|fieldName:Content`\n```yaml\n!be t:My title|d:My␣
↪description|thumb:https://i.imgur.com/010ZBCm.png|image:https://i.imgur.com/010ZBCm.
↪png|f:My footer|color:blurple|time:now\n```",
        "inline":
            false
    }],
    "title":
        "Build A Custom Embed",
    "description":
```

(continues on next page)

(continued from previous page)

```
        "Forget about the built-in `!embed` and `!cembed` commands, that are not␣
↪customizable nor very user friendly, as they require specific syntax, and the latter a␣
↪minimum knowledge of what JSON even is, which is not the average person.",
    "image":{
        "url":
            "https://i.imgur.com/SlCjHKv.png"
        },
        "color":3092790,
        "footer": {
            "text":"Custom command made with  for TagScript by Asty'#8926"
        }
    }
}

{=(JSONs):errJSON finalJSON}
{=(shouldErr):{in(true):{areAllFieldsEmpty} {isNoArg}}}

{=(JSONIdx):{{i}(true):! {shouldErr} true}}

{=(JSONToSend):{{JSONs({JSONIdx})}}}

{=(debug):__Debug:__
jargs: {jargs}

1: {w1} `{word1}` {content1}
2: {w2} `{word2}` {content2}
3: {w3} `{word3}` {content3}
4: {w4} `{word4}` {content4}
5: {w5} `{word5}` {content5}
6: {w6} `{word6}` {content6}
7: {w7} `{word7}` {content7}
8: {w8} `{word8}` {content8}
9: {w9} `{word9}` {content9}
10: {w10} `{word10}` {content10}
11: {w11} `{word11}` {content11}
12: {w12} `{word12}` {content12}

f.image: {f.image}
f.title: {f.name}
f.titleurl: {f.name}
f.name: {f.name}
f.nameicon: {f.nameicon}
f.nameurl: {f.nameurl}

**Color**
f.color: {f.color} - sepColor: {sepColor} - sanitizedColor: {sanitizedColor}
spacedHex: `{spacedHex}`
finalHexColor: `{finalHexColor}`
finalDecimalColor: `{finalDecimalColor}`

f.description: {f.description}
f.thumbnail: {f.thumbnail}
```

(continues on next page)

(continued from previous page)

~~~
f.footer: {f.footer}
f.footericon: {f.footericon}

f.timestamp: {f.timestamp}
}

{=(testing):Example:```css
{example}```
FINAL JSON:```json
{finalJSON}
```}

{=(debug2):__Debug 2:__
areAllFieldsEmpty: {areAllFieldsEmpty}
isNoArg: {isNoArg}

JSONToSend:```json
{JSONToSend}
```
}
{c:cembed {channelToSend} {JSONToSend}}
{override}
{require({requiredRolesErrorMessage} ):{requiredRoles}}
~~~

Tag Import

CHAPTER TWENTYFOUR

NILOC#0421

Table of Tags

24.1 Dictionary Tag With Image or Link Output

What:

A tag that gives the user a definition of a word

How:

This tag takes the user input and sends it to an API or application programming interface that I built using node.js and express. Once there it takes the word imputed and sends it to another API to get the definition, part of speech, etc.. Then using that info it either creates an image or a website with link previews depending on what is selected in the tag to output to the user. The website is using html meta tags to generate the preview but also is generating more info found on the actual website

Why:

I have seen other dictionary tags here on #show-off but they all require you to click a link and go to your browser in order to see the definition and other info. I wanted to make something that gave the user what they needed without ever having to leave discord.

Links:

Image of Tag

Image of Error Msgs

Image of Website

How to Customize:

To change if the tag uses an image or link preview all you have to do is change the `displayType` variable to "image" or "link". This is also explained in the tag comments

I would also like to mention this whole idea is possible and inspired by the stuff that **_Leg3ndary#5759** did in his shell tag and other post here about saving data with tagscript

Source Code

```
{=(COMMENT):--This tag can either  use a image or a link preview to display the␣
↪definition --}
{=(COMMENT):-- to change this just change the below displayType variable to "image" or
↪"link" --}
{=(displayType):link}

{=(COMMENT):--Don't touch anything from now on, unless you know what you're doing --}
{=():}
{{if({displayType}==link):blacklist(https://define-tag.niloc3.repl.co/l?word={1}&time=
↪{unix}):{server(id)}}}
```

Tag Import