YAGPDB

Custom Commands Examples

⚠ Note: This page is no longer updated with the latest versions of codes/commands. If you plan to copy paste codes for your server, see this website which is frequently updated with new community-contributed commands.

ⓘ This isn't the actual page about custom commands. A brief overview about custom commands can be found here. Please take notice, some of examples presented here are not up to date with current capabilities of YAGPDB. Please visit our support server for newer solutions.

Controlled randomizer example

YAGPDB has a built-in random response system for custom commands, but sometimes you may want to control the chances for certain responses to occur. You can do this by creating a singular response and creating a variable with randInt. Then use an if else if statement like this to print out your desired output.

```
{{$var := randInt 100}}

{{if lt $var 10}}
This has a 10% chance of being triggered
{{else if lt $var 35}}
This has a 25% chance of being triggered
{{else}}
This has a 65% chance of being triggered
{{end}}
```

Silent execution of commands or storage in a variable

This command is to be placed in the welcome message. It filters out people with invites in their name. Make sure that the checkbox **Censor server invites in usernames?** and the ban command are enabled on your server.

You might not want the response for the executed command to show. You can suppress the response of a command like the following:

Trigger type: `Join message in server channel`

```
{{if .UsernameHasInvite}}
{{$silent := execAdmin "ban" .User.ID "ad blocked"}}
{{else}}
```

```
{{/* Replace this with your normal join message or leave it as it is */}}
{{end}}
```

Range example

This command will teach you on how the range function works. It can iterate over many items including but not limited to a `cslice`, slice, `sdict`, and `dict`

This particular command loops over a cslice and a sdict.

Trigger type: `Command` Trigger: `range`

```
{{/* range can iterate over many things, let's start with slice */}}
{{ $slice := cslice "YAGPDB " "is " "cool!" }}
{{/* Here, we range over with 1 argument, meaning the dot will be set to current iteration
{{ range $slice -}}
    {{ . }}
{{- end -}}
{{ $map := sdict "foo" "bar" "hello" "world" }}
{{- /* Now, we range with two arguments - $k will be the KEY, $v will be VALUE (note dot i
{{- range $k, $v := $map }}
{{ $k }} - {{ $v }}
{{- end }}
```

`$k` is the index for arrays / cslices (starting at 0) or the key for maps and sdicts, while `$v` is the current word in your input that you are on.

Range will work on any kind of slice/array. for example. If we wanted to look for all the entries in our database we can use range and index through them all in the following.

```
{{$lb := dbTopEntries "%" 100 0}}
{{range $lb}}
{{.UserID}} **:** {{.Key}} **:** {{.Value}}
{{end}}
```

Note that we can go through everything that is in $lb with range.

Dictionary example

A dictionary does not currently have a lot of practical use, because YAGPDB has a data type more suited for most use cases - `sdict`. However, sdict only supports string keys, which means that in the case you want non-string keys, you will have to use `dict`.

Trigger type: `Command` Trigger: `dict`

```
{{ $dict := dict 0 "foobar" "hello" "world" }}
{{/* Retrieve value with integer key with index */}}
0 - {{ index $dict 0 -}}
{{/* Retrieve value with string key using dot notation */}}
```

```
hello - {{ $dict.hello }}
```

parseArgs example

The `parseArgs` template can check if specific arguments are given. If not, it will return a custom error message. It also checks if specific args are of a specific type and simplifies the argument management. Available types for `carg` are:

- `int` (whole number)
- `string` (text)
- `user` (user mentions as type user)
- `userid` (mentions or the user's ID, as integer)
- `channel` (channel mention or ID, as type channel)
- `role` (role name or ID, as type **discordgo.Role*)
- `duration` (duration as integer or string with optional time modifier - s,m,h, etc...)
- `member` (mentions or the user's ID, as type member)

Trigger type: `Command` Trigger: `send`

```
{{$args := parseArgs 2 "Syntax is <channel> <text>"
    (carg "channel" "channel to send to")
    (carg "string" "text to send")}}

{{sendMessage ($args.Get 0).ID ($args.Get 1)}}
```

Countdown example (Exec CC)

This example consists of two custom commands, and after copy/paste `REPLACE-WITH-...` arguments need to be replaced by actual custom command ID's in your system. This custom command is very complex, uses very many advanced functions, all it does, constructs a 10 second countdown timer command-system for given starting time.

```
{{$args := parseArgs 2 ""
  (carg "duration" "countdown-duration")
  (carg "string" "countdown-message")}}

{{$t := currentTime.Add ($args.Get 0)}}
{{$mID := sendMessageRetID nil (print "countdown starting... " $t.String)}}
{{execCC REPLACE-WITH-NEXT-CC-ID nil 0 (sdict "MessageID" $mID "T" $t "Message" ($args.Get
```

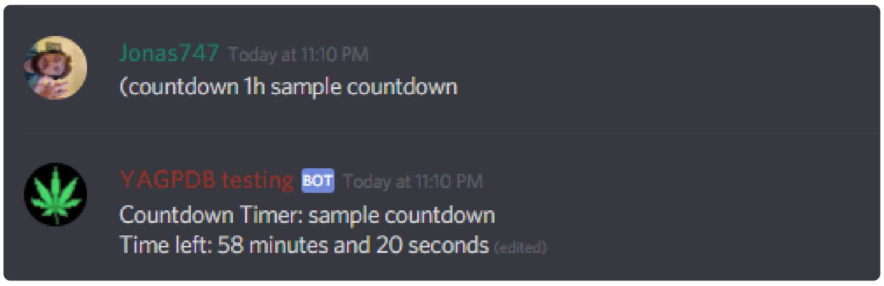
Second part of the custom commands, here we see, how `data`-part of exeCC was made in previous custom command as `sdict` and now we are calling those keys with `.ExecData` - for example `.ExecData.MessageID` sets new variable the same as stated in previous code.

```
{{$timeLeft := .ExecData.T.Sub currentTime}}
{{$cntDownMessageHeader := print "Countdown Timer: " .ExecData.Message}}
{{$formattedTimeLeft := humanizeDurationSeconds $timeLeft}}

{{$t := .ExecData.T}}
{{$mID := .ExecData.MessageID}}
{{$ts := .TimeSecond}}

{{if lt $timeLeft (mult .TimeSecond 30)}}
  {{range seq 1 (toInt $timeLeft.Seconds) }}
    {{$timeLeft := $t.Sub currentTime}}
    {{$formattedTimeLeft := humanizeDurationSeconds $timeLeft}}

    {{editMessage nil $mID (print $cntDownMessageHeader "\nTime left: " $formattedTimeLeft
    {{if gt $timeLeft $ts}} {{sleep 1}} {{end}}
  {{end}}
  {{editMessage nil  .ExecData.MessageID (print $cntDownMessageHeader "\nTime left: **ENDEI
{{else}}
    {{editMessage nil .ExecData.MessageID (print $cntDownMessageHeader "\nTime left: " $fo
    {{execCC .CCID nil 10 .ExecData}}
{{end}}
```


Jonas747 Today at 11:10 PM
(countdown 1h sample countdown

YAGPDB testing BOT Today at 11:10 PM
Countdown Timer: sample countdown
Time left: 58 minutes and 20 seconds (edited)

Jonas using YAGPDB testing bot here, same execCC custom commands.

Database example

This is a simple note taking system containing 3 separate custom commands. Also note that the actual name of the key inserted to database begins with "notes_".

Save note:

```
{{$args := parseArgs 2 ""
  (carg "string" "key")
  (carg "string" "value")}}

{{dbSet .User.ID (print "notes_" ($args.Get 0)) ($args.Get 1)}}
Saved `{{$args.Get 0}}` as `{{$args.Get 1}}`
```

Get note:

```
{{$key := print "notes_"  .StrippedMsg}}
```

```
{{$note := dbGet .User.ID $key}}
{{if $note}}

{{$strippedKey := slice $key 6 (len $key)}}
Note: `{{$strippedKey}}` Created {{humanizeTimeSinceDays $note.CreatedAt}} ago:
{{$note.Value}}

{{else}}Couldn't find any note like that :({{end}}
```

List user's notes:

```
{{$notes := dbGetPattern .User.ID "notes_%" 100 0}}
{{range $notes}}
{{- $strippedKey := slice .Key 6 (len .Key)}}
`{{$strippedKey}}` created {{humanizeTimeSinceDays .CreatedAt}} ago
{{- else}}
You don't have any notes :(
{{end}}
```

Cooldown Example

With YAGPDB's database system, you can now add cooldowns to you custom commands. You can either make them global cooldowns or a per user cooldown.

```
{{/* CONFIGURATION HERE CHANGE VALUES AS NEEDED */}}

{{/* 0 for per user, 1 for global */}}
{{$isGlobal := 1}}
{{/* name your cooldown name (anything works) */}}
{{$name := "replace with name here"}}
{{/* Length of the cooldown (in seconds) */}}
{{$lengthSec := 10}}

{{/* CREATING VARIABLES DO NOT TOUCH */}}
{{$id := 0}}
{{$key := print "cooldown_" $name}}
{{if eq $isGlobal 0}}
{{$id = .User.ID}}
{{end}}


{{if dbGet (toInt64 $id) $key}}
{{/* Code to execute when cooldown is active */}}
{{else}}
{{/* Create cooldown entry */}}
{{dbSetExpire (toInt64 $id) $key "cooldown" $lengthSec}}

{{/* YOUR COMMAND HERE */}}
{{end}}
```

User submitted custom commands

Counter Command

> By **Timcampy#5636**

With YAGPDB's database system, I made a command to have users count from 0 and keep counting to the next number. Relatively simple command that involves database and type conversion.

Trigger type: `Regex` Trigger: `\A`

`BE SURE TO RESTRICT THE COMMAND TO A SINGLE CHANNEL`

```
{{/* If you are not doing (no twice msg in a row)  or (role assignment for latest user)  y

{{/* First time running command, set up initial values*/}}
{{$lastUser := dbGet 118 "counter_user"}}
{{if $lastUser}}
{{else}}
{{dbSet 118 "counter_user" 0}}
{{dbSet 118 "counter_count" "0"}}
{{end}}

{{/* OPTIONAL: this is just to prevent one person to type all the numbers themselves */}}
{{/* If current user ID matches the user who last successfully ran command */}}
{{if eq (toFloat $lastUser.Value) (toFloat .User.ID)}}
{{deleteTrigger 0}}
{{sendDM "You can not send a msg twice in a row"}}
{{else}}


{{$next := dbGet 118 "counter_count"}}

{{/* If message is equal to the expected next number , update counter */}}
{{if eq (toInt .StrippedMsg) (toInt ($next.Value))}}
{{dbSet 118 "counter_count" (add (toInt ($next.Value)) 1)}}
{{$name := (add (toInt ($next.Value)) 1)}}
{{editChannelName .Channel.ID (joinStr "" "count-to-" $name )}}
{{/* OPTIONAL count tracker per user, Delete if you don't want to use */}}
{{$key := joinStr "" "counter_tracker_"  .User.ID}}
{{$userCount := dbGet 118 $key}}
{{if $userCount}}
{{dbSet 118 $key (add (toInt ($userCount.Value)) 1)}}
{{else}}
{{dbSet 118 $key 1}}
```

```
{{end}}

{{/* OPTIONAL: If you don't want to give a role to the latest person delete everything but
{{/* Give new user role, take role back from old user and update latest user */}}
{{/* (UPDATE THE ROLEID) */}}
{{giveRoleID .User.ID 606891664396648474}}
{{$tmpUser := (userArg (toInt $lastUser.Value))}}
{{/* check if its a valid user or not */}}
{{if $tmpUser}}
{{takeRoleID ($tmpUser.ID) 606891664396648474}}
{{end}}
{{dbSet 118 "counter_user" (toString .User.ID)}}
{{else}}

{{/* Message did not match expected next value */}}
{{deleteTrigger 0}}
{{/* Removed Because too annoying :^) */}}
{{/*sendDM "That is not the next number, learn how to count :)"*/}}
{{end}}
{{end}}
```

GiveRole command for specific roles

> By **GryTrean#8957**

This command will allow you to give a role to someone, making sure that the role given is in a list of allowed roles. We use the `{{giveRoleName <user> <role>}}` template which allows us to give a user a role by name. We also make sure that the command has the correct number of arguments and if not, we give a response with the correct usage of the command. To add a new exception to the roles that can be given, you simply add another role in line 2. You could also make the command take away roles from someone instead of giving them by simply using the `{{takeRoleName}}` template instead of `{{giveRoleName}}`.

Trigger type: `Command` Trigger: `giveRoleName`

GiveRole Custom Command

```
 1 {{if eq (len .Args) 3}}
 2     {{$allowedRoles := (cslice "Patron" "Quality Patron" "Paypal Donors")}}
 3     {{$role := (index .CmdArgs 1)}}
 4 
 5     {{if in $allowedRoles $role}}
 6         {{giveRoleName (userArg (index .CmdArgs 0)) $role}}
 7         Gave {{$role}} to {{index .CmdArgs 0}}! :white_check_mark:
 8     {{else}}
 9         You can't use this command to give that role to someone! :x:
10     {{end}}
11 {{else}}
12     Correct usage of the command: -giverole <target> "<rolename>"
13 {{end}}
```

Broadcast command

> By **GryTrean#8957**
> Updated by: **Timcampy#5636**

This command lets the bot send a message to another channel. It uses embeds so you can see `sdict` (dictionary but with only string keys), `sendMessage`, and `cembed` in action.

Trigger type: `Command` Trigger: `bc`

```
{{if eq (len .Args) 3}}
    {{$channel := (index .CmdArgs 0)}}
    {{$msg:= (joinStr " " (slice .CmdArgs 1))}}

    {{$footer1 := (sdict "text" (joinStr "" "This broadcast was sent by " (.User.Username)
    {{$msgEmbed := cembed "title" "Broadcast!" "description" ($msg) "color" 16763904 "foot

    {{sendMessage $channel $msgEmbed}}

    {{$desc := (joinStr "" "User " (.User.Username) " broadcasted a message in #" ($channe
    {{$footer := (sdict "text" (joinStr "" "The broadcast was sent on " (exec "ctime") "")

    {{$embed := cembed "title" "Broadcast sent!" "description" ($desc) "color" 4325120 "fo

    {{sendMessage nil $embed}}

{{else}}
    Correct usage of the command: -bc "<channel-name>" "<message>"
{{end}}
```

Avatar command

> By: **L-z#7749**

This command does a good job at using a little bit of everything. Which include but is not limited to, `conditional statement`, `assigning values to variable`, `getting command arguments`, `using template code`, and `creating embeds`. If you are able to understand everything in this command, you are at a very good place in being able to make advanced custom commands.

Trigger type: `Command` Trigger: `avatar`

```
{{$ln := (len .Args)}}
{{$sizes := (cslice "16" "32" "64" "128" "256" "512" "1024" "2048" "4096")}}
{{$err1 := "Wrong image size input format! Possible values: 16, 32, 64, 128, 256, 512, 102
{{$err2 := "Unknown user :("}}
{{$color := 1478046}}
{{if gt $ln 1}}
  {{$1 := (index .Args 1)}}
  {{if ($user := userArg $1)}}
    {{if gt $ln 2}}
      {{$2 := (index .Args 2)}}
      {{if in $sizes $2}}
        {{$out := $user.AvatarURL (toString $2)}}
        {{$emb := cembed "color" $color "image" (sdict "url" $out)}}
        {{sendMessage nil $emb}}
      {{else}}
        {{$err1}}
      {{end}}
    {{else}}
      {{$out := $user.AvatarURL "512"}}
      {{$emb := cembed "color" $color "image" (sdict "url" $out)}}
      {{sendMessage nil $emb}}
    {{end}}
  {{else if gt $ln 2}}
    {{$2 := (index .Args 2)}}
    {{if ($user := userArg $2)}}
      {{if in $sizes $1}}
        {{$out := $user.AvatarURL (toString $1)}}
        {{$emb := cembed "color" $color "image" (sdict "url" $out)}}
        {{sendMessage nil $emb}}
      {{else}}
        {{$err1}}
      {{end}}
    {{else}}
      {{$err2}}
    {{end}}
  {{else if in $sizes $1}}
    {{$out := .User.AvatarURL (toString $1)}}
    {{$emb := cembed "color" $color "image" (sdict "url" $out)}}
    {{sendMessage nil $emb}}
  {{else}}
    {{$err1}}
  {{end}}
{{else}}
  {{$out := .User.AvatarURL "512"}}
  {{$emb := cembed "color" $color "image" (sdict "url" $out)}}
  {{sendMessage nil $emb}}
{{end}}
```

Suggestion command

> By: **Michdi#1602**

This command is used to replace suggestion bots. You can adapt it to your needs.

Trigger type: `Command` Trigger: `suggest`

```
{{ $channel := 476178740133494784 }} {{/* Replace this with your suggestion channel ID */}

{{if gt (len .Args) 1}}
Suggestion submitted.
{{ $embed := cembed
"description" (joinStr " " .CmdArgs)
"color" 9021952
"author" (sdict "name" (joinStr "" .User.Username "#" .User.Discriminator) "url" "" "icon_
"timestamp"  currentTime
}}
{{ $id := (sendMessageNoEscapeRetID $channel $embed) }}
{{ addMessageReactions $channel $id "upvote:524907425531428864" "downvote:52490742503217563
{{else}}
Correct usage: `-suggest <suggestion>`
{{end}}
{{deleteResponse 5}}
{{deleteTrigger 5}}
```

Big emote command

> By: **CHamburr#2591**
> Updated by: **Joe_#2447**

This command uses the `reFindAllSubmatches` template as well as the `printf` template, and will enlarge custom emotes, whether still or animated. This will also work for emotes that are from the servers YAGPDB is not in, as it gets the emote file directly from Discord's database.

Trigger type: `Command` Trigger: `bigemote`

```
{{ $matches := reFindAllSubmatches `<(a)?:.*?:(\d+)>` .StrippedMsg }}
{{ if $matches }}
        {{ $animated := index $matches 0 1 }}
        {{ $id := index $matches 0 2 }}
        {{ $ext := ".png" }}
        {{ if $animated }} {{ $ext = ".gif" }} {{ end }}
        {{ $url := printf "https://cdn.discordapp.com/emojis/%s%s" $id $ext }}
        {{ sendMessage nil (cembed
                "title" "❯ Big Emoji"
                "url" $url
                "color" 14232643
                "image" (sdict "url" $url)
                "footer" (sdict "text" (joinStr "" "Emoji ID: " $id))
        ) }}
{{ else }}
        **Usage:** `-bigemoji <custom emoji>`.
{{ end }}
```