YAGPDB Functions

Template functions reference — with GroundPower/yagpdb-selfhost fork additions integrated in place

SELF-HOST EDITION

Functions are used to take action within template scripts. Some functions accept arguments, and some functions return values you can send in your response or use as arguments for other functions.

HTTP

> **Note: Self-Host Addition**
> These functions are additions in the self-hosted fork (GroundPower/yagpdb-selfhost) and are not part of upstream YAGPDB.

These functions perform outbound HTTP GET requests from within a custom command.

> **Caution: Limits & Security**
> 5-second timeout, 1 MB maximum response body, `http` / `https` schemes only. Requests are SSRF-protected: the target host is resolved and connections to private, loopback, CGNAT, link-local (including cloud metadata `169.254.169.254`), and IPv6 ULA addresses are refused — so a custom command cannot reach Docker-internal services such as the bot's own database or Redis. There is no URL allow-list; any public URL is permitted, filtered only by the SSRF guard.

> **Note: Rate Limit**
> `httpGet` and `httpGetJSON` share a combined limit of 10 calls per custom command execution (60 on premium), counter `http_get`. Requests are sent with `User-Agent: YAGPDB-selfhost/1.0 (httpGet template fn)`.

`httpGet`

```
{{ $body := httpGet <url> }}
```

Performs an HTTP GET request to `url` and returns the response body as a string. Returns an error if the response status is 400 or greater, if the body exceeds 1 MB, or if the URL fails validation (scheme / host / SSRF). The returned string is additionally bounded by the maximum string length.

`httpGetJSON`

```
{{ $data := httpGetJSON <url> }}
```

Like `httpGet`, but parses the response body as JSON and returns a navigable value: an sdict for JSON objects and a slice for JSON arrays, so you can traverse it with `.Field` access or the `index` function. Returns an error on a status of 400 or greater, or on invalid JSON.

```
{{ $data := httpGetJSON "https://example.com/champions.json" }}
{{ $champ := index $data.champions "aatrox" }}
{{ $champ.name }}
```

Channel

These functions relate to channels and threads.

> **Tip: Current Channel or Thread**
> Unless specified otherwise, these functions accept an ID, name, or `nil` for their thread or channel argument.

`addThreadMember`

```
{{ addThreadMember <thread> <member> }}
```

Adds a member to an existing thread. Does nothing if either argument is invalid.

`closeThread`

```
{{ closeThread <thread> <lock> }}
```

Closes or locks the given thread. You cannot lock a closed thread, but you can close a locked thread.

- `lock`: whether to lock the thread instead. Default `false`.

createForumPost

```
{{ $post := createForumPost <channel> <name> <content> [values] }}
```

Creates a new forum post. Returns a channel object on success.

- `channel`: the forum channel to post to.
- `name`: The post title. May not be empty. Must be a string.
- `content`: the initial message's content; may be a string, an embed, or a complex message. May not be empty.
- `values` (optional): Additional options for the post. May include:
  - `"slowmode"`: The thread's slowmode in seconds.
  - `"tags"`: One or more forum tag name or ID. Duplicate and invalid tags are ignored.

createThread

```
{{ $thread := createThread <channel> <messageID> <name> [private] [auto_archive_duration] [invitable] }}
```

Creates a new thread in the specified channel. Returns a channel object on success.

- `channel`: the parent channel to create the thread in.
- `message`: either `nil` to create a channel thread, or a message ID to create a message thread.
- `private`: whether the thread is private. Default `false`.
- `auto_archive_duration`: how long the thread will show in the channel list after inactivity. Valid values are 60, 1440, 4320, and 10080 minutes. Defaults to 10080 (7 days).
- `invitable`: whether non-moderators can add other members to the thread. (true/false)

Note: There is no functional difference between a channel thread and a message thread.

Because the optional arguments are positional, you must provide the preceding ones if you wish to override a later option. Consider the following example to create a public thread in the current channel with no message reference that is archived after an hour and allows non-moderators to add others:

```
{{ createThread nil nil "new thread" false 60 true }}
```

deleteForumPost

```
{{ deleteForumPost <post> }}
```

Deletes the given forum post.

This function is functionally the same to deleteThread. Use whichever function is semantically more meaningful in the context of your custom command.

deleteThread

```
{{ deleteThread <thread> }}
```

Deletes the given thread.

This function is functionally the same to deleteForumPost. Use whichever function is semantically more meaningful in the context of your custom command.

editChannelName

```
{{ editChannelName <channel> <newName> }}
```

Edits the name of the given channel.

- `newName`: the new name for the channel. Must be a string.

This function is, together with editChannelTopic, limited to 10 calls per custom command execution. In addition to this, Discord limits the number of channel modifications to 2 per 10 minutes.

editChannelTopic

```
{{ editChannelTopic <channel> <newTopic> }}
```

Edits the topic of the given channel.

- `newTopic`: the channel's new topic. Must be a string. Discord markdown is supported.

This function is, together with editChannelName, limited to 10 calls per custom command execution. In addition to this, Discord limits the number of channel modifications to 2 per 10 minutes.

editThread

```
{{ editThread <thread> <opts> }}
```

Edits the specified thread.

- `opts`: a sdict containing the thread parameters to edit, supporting the following keys (all optional):
  - `slowmode`: the thread's slowmode in seconds.
  - `tags`: one or more forum tag name or ID. Duplicate and invalid tags are ignored.
  - `auto_archive_duration`: how long the thread will show in the channel list after inactivity.
  - `invitable`: whether non-moderators can add other members to the thread. Defaults to false.

getChannelOrThread

```
{{ $channel := getChannelOrThread <channel> }}
```

Returns the full channel or thread object for the given channel.

getChannelPins

```
{{ $pins := getChannelPins <channel> }}
```

Returns a slice of message objects pinned to the given channel or thread.

Rate-limited to 2 (premium: 4) calls per custom command execution.

getChannel

```
{{ $channel := getChannel <channel> }}
```

Returns the full channel object for the given channel. Will not work for threads.

getPinCount

```
{{ $numPins := getPinCount <channel> }}
```

Returns the number of pinned messages in given channel.

getThread

```
{{ $thread := getThread <thread> }}
```

Returns the full thread object for the given thread. Will not work for channels.

openThread

```
{{ openThread <thread> }}
```

Reopens the given thread.

pinForumPost

```
{{ pinForumPost <post> }}
```

Pins the given forum post, which may be specified by its ID or name.

removeThreadMember

```
{{ removeThreadMember <thread> <member> }}
```

Removes the given member from the given thread.

unpinForumPost

```
{{ unpinForumPost <post> }}
```

Unpins the given forum post, which may be specified by its ID or name.

Database

These functions help you interact with the custom command database.

dbBottomEntries

```
{{ $entries := dbBottomEntries <pattern> <amount> <nSkip> }}
```

Returns up to `amount` entries from the database, sorted in ascending order by the numeric value, then by entry ID.

- `amount`: the maximum number of entries to return, capped at 100.
- `pattern`: the PostgreSQL pattern to match entries against.
- `nSkip`: the number of entries to skip before returning results.

dbCount

```
{{ $count := dbCount <userID|pattern|query> }}
```

Returns the count of all matching database entries that are not expired.

The argument must be one of the following:

- `userID`: count entries for the given user ID.
- `pattern`: count only entries with keys matching the given pattern.
- `query`: an sdict with the following (all optional) keys:
  - `userID`: only count entries with a matching UserID field. Defaults to all UserIDs.
  - `pattern`: only counts entries with keys matching the given pattern. Defaults to all keys.

dbDelByID

```
{{ dbDelByID <userID> <ID> }}
```

Deletes a database entry under the given `userID` by its `ID`.

dbDelMultiple

```
{{ $numDeleted := dbDelMultiple <query> <amount> <nSkip> }}
```

Deletes up to `amount` entries from the database matching the given criteria. Returns the number of deleted entries.

- `query`: an sdict with the following (all optional) keys:
  - `userID`: only delete entries with a matching UserID field. Defaults to all UserIDs.
  - `pattern`: only delete entries with keys matching the given pattern. Defaults to all keys.

    - `reverse`: whether to delete entries with the lowest value first. Default is `false` (highest value first).
- `amount`: the maximum number of entries to delete, capped at 100.
- `nSkip`: the number of entries to skip before deleting.

dbDel

```
{{ dbDel <userID> <key> }}
```

Deletes the specified entry from the database.

dbGetPatternReverse

```
{{ $entries := dbGetPatternReverse <userID> <pattern> <amount> <nSkip> }}
```

Retrieves up to `amount` entries from the database in descending order as a slice.

- `userID`: the user ID to retrieve entries for.
- `pattern`: the PostgreSQL pattern to match entries against.
- `amount`: the maximum number of entries to return, capped at 100.
- `nSkip`: the number of entries to skip before returning results.

See dbGetPattern for a function that retrieves entries in ascending order.

dbGetPattern

```
{{ $entries := dbGetPattern <userID> <pattern> <amount> <nSkip> }}
```

Returns up to `amount` entries from the database in ascending order as a slice.

- `userID`: the user ID to retrieve entries for.
- `pattern`: the PostgreSQL pattern to match entries against.
- `amount`: the maximum number of entries to return, capped at 100.
- `nSkip`: the number of entries to skip before returning results.

See dbGetPatternReverse for a function that retrieves entries in descending order.

dbGet

```
{{ $entry := dbGet <userID> <key> }}
```

Returns the specified database entry.

dbIncr

```
{{ $newValue := dbIncr <userID> <key> <incrBy> }}
```

Increments the value of the specified database entry by `incrBy`. Returns the new value as a floating-point number.

- `incrBy`: the amount to increment the value by. Must be a valid number.

dbRank

```
{{ $rank := dbRank <query> <userID> <key> }}
```

Returns the rank of the specified entry in the set of entries as defined by `query`.

- `query`: an sdict with the following (all optional) keys:
    - `userID`: only include entries with the given user ID.
    - `pattern`: only include entries with keys matching the given pattern.
    - `reverse`: if `true`, entries with lower values have higher ranks. Default is `false`.

dbSetExpire

```
{{ dbSetExpire <userID> <key> <value> <ttl> }}
```

Same as dbSet but with an additional expiration `ttl` in seconds.

dbSet

```
{{ dbSet <userID> <key> <value> }}
```

Sets the value for the specified `key` and `userID` to `value`.

- `value`: an arbitrary value to set.

dbTopEntries

```
{{ $entries := dbTopEntries <pattern> <amount> <nSkip> }}
```

Returns up to `amount` entries from the database, sorted in descending order by the numeric value, then by entry ID.

- `pattern`: the PostgreSQL pattern to match entries against.
- `amount`: the maximum number of entries to return, capped at 100.
- `nSkip`: the number of entries to skip before returning results.

> **Caution: Storing Numerical Values**
>
> Numerical values are stored as floating-point numbers in the database; large numbers such as user IDs will lose precision. To avoid this, convert them to a string before writing to the database.
>
> Because numerical `dict` keys are retrieved as an `int64`, you have to first convert to `int64` when retrieving such values. So, to retrieve the value associated with the numerical key `N`: `{{ $dict.Get (toInt64 N)}}`.

Encoding and Decoding

decodeBase64

```
{{ $decoded := decodeBase64 <string> }}
```

Undoes the transformation performed by `encodeBase64`, converting the base64-encoded `string` back to its original form.

encodeBase64

```
{{ $encoded := encodeBase64 <string> }}
```

Encodes the input `string` to base64.

hash

```
{{ $hash := hash <string> }}
```

Generates the SHA256 hash of the input string.

json

```
{{ $json := json <value> [indent] }}
```

Encodes `value` as JSON. If the `indent` flag is `true`, the output is pretty-printed with appropriate indentation.

jsonToSdict

```
{{ $sdict := jsonToSdict <json> }}
```

Parses the JSON-encoded data into a string-dictionary, returning an error if the input was invalid JSON.

urlescape

```
{{ $result := urlescape <string> }}
```

Escapes the input `string` such that it can be safely placed inside a URL path segment, replacing special characters (including `/`) with `%XX` sequences as needed.

urlunescape

```
{{ $result := urlunescape <string> }}
```

Undos the transformation performed by `urlescape`, converting encoded substrings of the form `%AB` to the byte `0xAB`.

urlquery

```
{{ $result := urlquery <string> }}
```

Returns the escaped value of the textual representation of the arguments in a form suitable for embedding in a URL query.

Executing Custom Commands

These functions enable you to execute a custom command within an already running custom command.

cancelScheduledUniqueCC

```
{{ cancelScheduledUniqueCC <ccID> <key> }}
```

Cancels a previously scheduled custom command execution using scheduleUniqueCC.

execCC

```
{{ execCC <ccID> <channel> <delay> <data> }}
```

Executes another custom command specified by `ccID`.

- `ccID`: the ID of the custom command to execute.
- `channel`: the channel to execute the custom command in. May be `nil`, a channel ID, or a channel name.
- `delay`: the delay in seconds before executing the custom command.
- `data`: some arbitrary data to pass to the executed custom command.

Calling `execCC` with 0 delay sets `.StackDepth` to the current recursion depth and limits it to 2. `execCC` is rate-limited strictly to a maximum of 10 delayed custom commands executed per channel per minute. Executions beyond this number will be dropped.

Example

The following example showcases a custom command executing itself.

```
{{ if .ExecData }}
  {{ sendMessage nil (print "Executing custom command ... Got data: " .ExecData) }}
  {{ return }}
{{ end }}

{{ sendMessage nil "Starting up ... " }}
{{ execCC .CCID nil 5 "Hello, world!" }}
```

scheduleUniqueCC

```
{{ scheduleUniqueCC <ccID> <channel> <delay> <key> <data> }}
```

Schedules a custom command execution to occur in the future, identified by `key`.

- `ccID`: the ID of the custom command to execute.

- `channel`: the channel to execute the custom command in. May be `nil`, a channel ID, or a channel name.
- `delay`: the delay in seconds before executing the custom command.
- `key`: a unique key to identify the scheduled custom command.
- `data`: some arbitrary data to pass to the executed custom command.

To cancel such a scheduled custom command before it runs, use cancelScheduledUniqueCC.

Interactions

Use of interactions within YAGPDB is an advanced topic; the documentation should be used only as reference. To learn about using interactions, see here.

Interaction Responses

- Only one interaction response may be sent to each interaction.
- If you do not send an interaction response, members will see "This application did not respond" on Discord.
- You may only send an interaction response to the interaction which triggered the command.
- Text output directly to the response is automatically sent as an interaction response if the interaction hasn't already been responded to.
- A CC executed with `execCC` by the triggered CC will be able to send initial responses to the triggering interaction.
- A response is not the same thing as a followup.

sendModal

```
{{ sendModal <modal> }}
```

Sends a modal to the member who triggered the interaction.

- `modal`: an sdict with the following keys:
  - `title`: the title of the modal.
  - `custom_id`: a unique identifier for the modal.
  - `fields`: a slice of sdicts with the following keys:
    - `label`: the label for the field.
    - `placeholder`: the placeholder text for the field.
    - `value`: the default value for the field.
    - `required`: whether the field is required.
    - `style`: the style of the field (1 for short, 2 for long).
    - `min_length`: the minimum length of the field.
    - `max_length`: the maximum length of the field.

Alternatively, you can create a modal object using the `cmodal` function.

Example

```
{{ $modal := sdict
  "title" "My Custom Modal"
  "custom_id" "modals-my_first_modal"
  "fields" (cslice
    (sdict "label" "Name" "placeholder" "Duck" "required" true)
    (sdict "label" "Do you like ducks?" "value" "Heck no")
    (sdict "label" "Duck hate essay" "min_length" 100 "style")) }}
{{ sendModal $modal }}
```

updateMessage

```
{{ updateMessage <newMessage> }}
```

Edits the message on which the button, select menu, or modal was triggered on.

- `newMessage`: the new message content. May be a string, an embed, or a complex message edit.

Example

The following example must be triggered by a component or modal submission.

```
{{ $button := cbutton "label" "I won!" "custom_id" "i_won" }}
{{ $content := printf "Press this button when you win! The last person who won was %s! They wanted to say they are a %s
{{ $message := complexMessageEdit "content" $content "buttons" $button }}
{{ updateMessage $message }}
```

updateMessageNoEscape

```
{{ updateMessageNoEscape <newMessage> }}
```

Same as updateMessage, plus it does not escape mentions.

Interaction Followups

- Interaction followups may be sent up to 15 minutes after an interaction.
- To send a followup, you must have the interaction token of the interaction you are following up.
- You can send as many followups as you'd like.
- Text output directly to the response is automatically sent as an interaction followup if the interaction has already been responded to.
- A followup is not the same thing as a response.

editResponse

```
{{ editResponse <interactionToken> <messageID> <newContent> }}
```

Edits a response to an interaction.

- `interactionToken`: the token of the interaction to edit. `nil` for the triggering interaction.
- `messageID`: the ID of a follow-up message. `nil` for the original interaction response.
- `newContent`: the new content for the message.

Example

The following example must be triggered by a component trigger or modal submission.

```
{{ $token := .Interaction.Token }}

{{ sendResponse nil "Here's the first message!" }}
{{ $id := sendResponseRetID $token (complexMessage "content" "Here's a sneaky one!" "ephemeral" true) }}

{{ sleep 2 }}

{{ editResponse $token $id (print "I've edited this message to say " noun) }}
{{ $editedResponse := getResponse $token $id }}
{{ editResponse $token nil $editedResponse.Content }}
```

editResponseNoEscape

```
{{ editResponseNoEscape <interactionToken> <messageID> <newContent> }}
```

Same as editResponse, plus it does not escape mentions.

Interaction Response/Followup Hybrids

Hybrid functions will send an interaction response if the interaction has not already been responded to, otherwise they will send the equivalent followup function. See editResponse for an example using `sendResponse*` functions.

sendResponse

```
{{ sendResponse <interactionToken> <message> }}
```

Sends a message in response to an interaction. Supports the `ephemeral` flag in `complexMessage`.

sendResponseNoEscape

```
{{ sendResponseNoEscape <interactionToken> <message> }}
```

Same as sendResponse, plus it does not escape mentions.

sendResponseNoEscapeRetID

```
{{ sendResponseNoEscapeRetID <interactionToken> <message> }}
```

Same as sendResponseNoEscape, but also returns the message ID.

sendResponseRetID

```
{{ sendResponseRetID <interactionToken> <message> }}
```

Same as sendResponse, but also returns the message ID.

Interaction Miscellaneous

cbutton

```
{{ $button := cbutton "list of button values" }}
```

Creates a button object for use in interactions.

A link style button *must* have a URL and may not have a Custom ID. All other styles *must* have a Custom ID and cannot have a URL. All buttons must have either a label or an emoji.

Example

```
{{ $button := cbutton "label" "Button" "custom_id" "buttons-duck" }}
{{ $message := complexMessage "buttons" $button }}
{{ sendMessage nil $message }}
```

cmenu

```
{{ $menu := cmenu "list of select menu values" }}
```

Creates a select menu object for use in interactions.

The type should be provided as a string: `"text"`, `"user"`, `"role"`, `"mentionable"`, or `"channel"`. Text type menus *must* have `options`, while all other types cannot.

Example

```
{{ $menu := cmenu
  "type" "text"
  "placeholder" "Choose a terrible thing"
  "custom_id" "menus-duck"
  "options" (cslice
    (sdict "label" "Two Ducks" "value" "opt-1" "default" true)
    (sdict "label" "A Duck" "value" "duck-option" "emoji" (sdict "name" "🦆"))
    (sdict "label" "Half a Duck" "value" "third-option" "description" "Don't let the smaller amount fool you."))
  "max_values" 3 }}

{{ sendMessage nil (complexMessage "menus" $menu) }}
```

cmodal

```
{{ $modal := cmodal "list of modal values" }}
```

Creates a modal object for use in interactions. See `sendModal` for more detail.

deleteInteractionResponse

```
{{ deleteInteractionResponse <interactionToken> <messageID> [delay] }}
```

Deletes the specified response or follow-up message.

- `interactionToken` : a valid interaction token or `nil` for the triggering interaction.
- `messageID` : valid message ID of a follow-up, or `nil` for the original interaction response.
- `delay` : an optional delay in seconds, max 10 seconds. Default: 10 seconds.

If you require a delay of more than 10 seconds, consider using `execCC` for deletion of an ephemeral response, or `deleteMessage` to delete a regular interaction response.

ephemeralResponse

```
{{ ephemeralResponse }}
```

Tells the bot to send the response text as an ephemeral message. Only works when triggered by an interaction. Works on responses and follow-ups.

Example

```
{{ ephemeralResponse }}

This text is invisible to others!
```

getResponse

```
{{ $response := getResponse <interactionToken> <messageID> }}
```

Returns the response or follow-up with the specified message ID belonging to the given interaction as a message object. Is also valid for ephemeral messages.

Components V2

Components V2 provides a new way to create interactive and visually appealing message layouts in Discord applications, making it easier to control message formatting and user interaction while maintaining line length under 120 characters.

componentBuilder

A `componentBuilder` simplifies building Discord's ComponentsV2 in custom commands.

```
{{ $component := componentBuilder (sdict [text] [section] [gallery] [file] [separator] [container] [buttons] [menus] [
```

Returns a complex message object with the given components.

All keys are optional, but the Discord API will reject completey empty messages, so some content is required.

- `text` : A string or a slice of strings.
- `section` : A layout block that shows text with one mandatory accessory: either a button or a thumbnail with the following keys:
  - `text` : A string or a slice of strings.
  - `button` : A button object.
  - `thumbnail` : An sdict with the following keys:
    - `media` : A string.
    - `description` : A string.
    - `spoiler` : A bool.
- `gallery` : Displays one or more media items with optional descriptions and spoiler flags with the following keys:
  - `media` : A string.
  - `description` : A string.
  - `spoiler` : A bool.
- `file` : Attaches text files to the message and optionally displays them with the following keys:

  - `content` : A string. (max 100 000 chars)
  - `name` : A string. (.txt appended automatically)
- `separator` : Adds spacing between components with the following keys:
  - `true` : large separator
  - `false` or `nil` : small separator
- `container` : Top-level layout. Containers offer the ability to visually encapsulate a collection of components, and have an optional customizable accent color bar.

Contains the following keys:

- `components` : A componentBuilder or a slice thereof.
- `color` : hex accent color (optional).
- `spoiler` : hides content until revealed (optional).
- `buttons` : Interactive buttons users can click. Can be single or multiple.
- `menus` : Interactive menuss users can select from. Can be single or multiple.
- `interactive_components` : Mix of buttons and menus, auto-distributed.
- `allowed_mentions` : A sdict with the following keys:
  - `users` : A slice of user IDs.
  - `roles` : A slice of role IDs.
  - `everyone` : A bool.
  - `replied_user` : A bool.
- `reply` : A sdict with the following keys:
  - `message_id` : A string.
  - `thread_id` : A string.
- `silent` : A bool.
- `ephemeral` : A bool.

Component Builder Functions

The `ComponentBuilder` simplifies building Discord's V2 components, allowing complex layouts to be built incrementally. It provides methods for constructing, manipulating, and exporting components in a format Discord understands, ensuring line length doesn't exceed 120 characters.

`ComponentBuilder.Add`

Adds a single component entry to the builder under the given key.

```
{{ $builder.Add <key> <value> }}
```

- `key` – The top-level key for the component (e.g., "text", "section", "buttons").
- `value` – The component data (string, sdict, Button, SelectMenu, etc.).

`ComponentBuilder.AddSlice`

Adds multiple components under one key.

```
{{ $builder.AddSlice <key> <values ... > }}
```

- `key` – The top-level key for the component (e.g., "text", "section", "buttons").
- `values` – The component data (string, sdict, Button, SelectMenu, etc.).

`ComponentBuilder.Merge`

Combine another builder into the current one.

```
{{ $builder.Merge <other> }}
```

- `other` – The other component builder to merge.

`ComponentBuilder.Get`

Returns the component data for the given key.

```
{{ $value := <builder>.Get <key> }}
```

- `key` – The top-level key for the component (e.g., "text", "section", "buttons").

Example usage can be found at the Components v2.

Modal Components

These functions aid you in creating components for use in modals, which are sent using `sendModal`.

`modalBuilder`

```
{{ $builder := modalBuilder }}
```

Returns a `modalBuilder`, which simplifies the construction of modal components.

You can optionally pass the following **positional** arguments to initialize the modal with the given values:

```
{{ $builder := modalBuilder title custom_id components ... }}
```

See also modalBuilder.Set.

modalBuilder.Set

```
{{ $builder.Set <key> <value> }}
```

Sets the value of a modal component field. `key` must be one of the following:

- `title`: the modal's title. Max 45 characters.
- `custom_id`: a unique identifier for the modal.
- `components`: a slice of modal components, created with below functions.

modalBuilder.AddComponents

```
{{ $builder.AddComponents (values ...) }}
```

Adds one or more modal components to the builder. `values` may be a single component or a slice of components.

In addition to the already existing components, you can also create modal-specific components using the functions outlined below.

> **Warning**
> All following components must be wrapped in a label to be used in a modal.

`ccheckbox`

```
{{ $checkbox := ccheckbox (values ...) }}
```

Returns a checkbox component for use in modals.

`values` may be an sdict or a list of key-value pairs with the following keys:

- `custom_id`: a unique identifier for the checkbox.
- `default`: optional bool for whether the checkbox is checked by default.

`ccheckboxGroup`

```
{{ $checkboxGroup := ccheckboxGroup (values ...) }}
```

Returns a checkbox group component for use in modals.

`values` may be an sdict or a list of key-value pairs with the following keys:

- `custom_id`: a unique identifier for the checkbox group.
- `min_values`: optionally the minimum number of checkboxes that must be checked. Min 0, max 10, defaults to 1. If 0, `required` must be false.
- `max_values`: optionally the maximum number of checkboxes that can be checked. Min 1, max the number of options, defaults to number of options.
- `required`: optional bool for whether selecting within the group is required.
- `options`: a slice of sdicts with the following keys:
  - `label`: User-facing label for the checkbox.

- `value`: the value that will be sent when the checkbox is checked.
- `description`: the optional description for the checkbox. Max 100 characters.
- `default`: optional bool for whether the checkbox is checked by default.

cradioGroup

```
{{ $radioGroup := cradioGroup (values ...) }}
```

Returns a radio group component for use in modals for selecting exactly one option from a defined list.

`values` may be an sdict or a list of key-value pairs with the following keys:

- `custom_id`: a unique identifier for the radio group.
- `required`: optional bool for whether selecting an option is required. Defaults to true.
- `options`: a slice of sdicts with the following keys:
  - `label`: User-facing label for the radio option.
  - `value`: the value that will be sent when the option is selected.
  - `description`: the optional description for the option. Max 100 characters.
  - `default`: optional bool for whether the option is selected by default. Only one option can be selected by default.

ctextInput

```
{{ $textInput := ctextInput (values ...) }}
```

Returns a text input component for use in modals.

`values` may be an sdict or a list of key-value pairs with the following keys:

- `custom_id`: a unique identifier for the text input.
- `style`: the style of the text input, either 1 for short, 2 for long.
- `min_length`: the optional minimum length of the input. Min 0, max 4000.
- `max_length`: the optional maximum length of the input. Min 1, max 4000.
- `required`: optional bool for whether the text input is required. Defaults to true.
- `value`: the optional default value for the text input. Max 4000 characters.
- `placeholder`: the optional placeholder text for the text input. Max 100 characters.

clabel

```
{{ $label := clabel (values ...) }}
```

Returns a label component for use in modals.

Labels are top-level components that wrap modal components.

`values` may be an sdict or a list of key-value pairs with the following keys:

- `label`: the label text. Max 45 characters.
- `component`: The component within.
- `description`: the optional label description. Max 100 characters.

---

ctextDisplay

```
{{ $textDisplay := ctextDisplay (values ...) }}
```

Creates a text display component.

`values` may be an sdict or a list of key-value pairs with the following keys:

- `content`: the text to display. Markdown supported.

Math

abs

```
{{ $result := abs x }}
```

Returns the absolute value of the provided number.

add

```
{{ $sum := add x y [...] }}
```

Returns the sum of the provided numbers. Detects the first number's type and performs the operation accordingly.

bitwiseAnd

```
{{ $result := bitwiseAnd x y [...] }}
```

Performs a bitwise AND operation on the provided numbers and returns the result.

bitwiseAndNot

```
{{ $result := bitwiseAndNot x y }}
```

Performs a bitwise AND NOT operation on the two provided numbers and returns the result.

bitwiseNot

```
{{ $result := bitwiseNot x }}
```

Performs a bitwise NOT operation on the provided number and returns the result.

bitwiseOr

```
{{ $result := bitwiseOr x y [...] }}
```

Performs a bitwise OR operation on the provided numbers and returns the result.

bitwiseXor

```
{{ $result := bitwiseXor x y }}
```

Performs a bitwise XOR operation on the two provided numbers and returns the result.

bitwiseLeftShift

```
{{ $result := bitwiseLeftShift x y }}
```

Shifts X left by Y bits and returns the result.

bitwiseRightShift

```
{{ $result := bitwiseRightShift x y }}
```

Shifts X right by Y bits and returns the result.

cbrt

```
{{ $result := cbrt x }}
```

Returns the cube root of the provided number.

div

```
{{ $result := div x y [...] }}
```

Performs division on the provided numbers. Detects the first number's type and performs the operation accordingly. If you need a floating-point number as a result of integer division, use fdiv.

fdiv

```
{{ $result := fdiv x y [...] }}
```

Special case of div; always returns a floating-point number as result.

log

```
{{ $result := log x [base] }}
```

Returns the logarithm of X with the given base. If no base is provided, the natural logarithm is used.

mathConst

```
{{ $result := mathConst "constant" }}
```

Returns the value of the specified math constant. See the math constants list.

max

```
{{ $result := max x y }}
```

Returns the larger of the two provided numbers.

min

```
{{ $result := min x y }}
```

Returns the smaller of the two provided numbers.

mod

```
{{ $result := mod x y }}
```

Returns the floating-point remainder of the division of X by Y.

Takes the sign of X, so `mod -5 3` results in `-2`, not `1`. To ensure a non-negative result, use `mod` twice: `{{ mod (add (mod x y) y) y }}`.

mult

```
{{ $result := mult x y [...] }}
```

Performs multiplication on the provided numbers. Detects the first number's type and returns the result accordingly.

pow

```
{{ $result := pow x y }}
```

Returns X raised to the power of Y as a floating-point number.

randInt

```
{{ $result := randInt [start] stop }}
```

Returns a random integer in the right-closed interval of `[0, stop)` or `[start, stop)` if two arguments are provided. That is, the result is always greater than or equal to `start` and strictly less than `stop`.

round

```
{{ $result := round x }}
```

Returns the nearest integer to X as float. Normal rounding rules apply.

roundCeil

```
{{ $result := roundCeil x }}
```

Returns the smallest integer greater than or equal to X. Put simply, always round up.

roundEven

```
{{ $result := roundEven x }}
```

Returns the nearest integer to X, rounding ties (x.5) to the nearest even integer.

roundFloor

```
{{ $result := roundFloor x }}
```

Returns the largest integer less than or equal to X. Put simply, always round down.

sqrt

```
{{ $result := sqrt x }}
```

Returns the square root of X as a floating-point number.

sub

```
{{ $result := sub x y [...] }}
```

Subtracts the provided numbers from each other. Detects the first number's type and returns the result accordingly.

Member

editNickname

```
{{ editNickname "newNick" }}
```

Edits the nickname of the member who triggered the command. The bot must have the `MANAGE_NICKNAMES` permission and be higher in the role hierarchy than the member. The bot cannot change the nickname of the server owner.

getMember

```
{{ $member := getMember <mention|userID> }}
```

Returns the member object for the given mention or user ID.

getGuildMembers

> **Note: Self-Host Addition**
> This function is an addition in the self-hosted fork (GroundPower/yagpdb-selfhost) and is not part of upstream YAGPDB.

```
{{ getGuildMembers [key value ...] }}
```

Fetches members of the current guild from the Discord API and returns a slice of members (`*discordgo.Member`). Called with no arguments it returns up to 25 members of the current guild. Arguments are passed as key/value pairs.

- `limit` : how many members to fetch. Default 25, minimum 1, maximum 10000.
- `after` : pagination — return members with an ID greater than this user ID.
- `guildid` : fetch from another guild instead of the current one (bot admin only).

Rate limited by counter `guild_getMembers` , cost 2 per call.

```
{{ $members := getGuildMembers "limit" 100 }}
{{ range $members }}{{ .User.Username }} {{ end }}
```

getMemberVoiceState

```
{{$state := getMemberVoiceState <member> }}
```

Returns the voice state for the member specified by ID or mention, or `nil` if the member is not in a voice channel.

> **Note: Voice Moderation (Self-Host Addition)**
> The five functions below are additions in the self-hosted fork (GroundPower/yagpdb-selfhost) and are not part of upstream YAGPDB. They let custom commands move members between voice channels, disconnect them, and server-mute / deafen them, complementing the read-only `getMemberVoiceState` .

> **Note: Bot Permissions**
> The bot must have the corresponding voice permission for each action: `MOVE_MEMBERS` for `moveMember` , `MUTE_MEMBERS` for `muteMember` / `unmuteMember` , and `DEAFEN_MEMBERS` for `deafenMember` / `undeafenMember` . The bot must also be higher in the role hierarchy than the target member.

> **Caution: Target Must Be Connected**
> All five functions require the target member to currently be connected to a voice channel. If they are not, Discord returns an error which is propagated to your custom command — wrap calls in `try` / `catch` to handle this gracefully.

> **Note: Rate Limit**
> These functions share a combined limit of 50 calls per custom command execution (counter `voice_mod` , cost 1 each). In addition, Discord enforces its own per-route rate limits on guild member modifications.

moveMember

```
{{ moveMember <target> <channel> }}
```

Moves a member to the specified voice channel. Pass `nil` or `0` as the channel argument to disconnect the member from voice instead of moving them.

- `target` : a user ID, mention, or user object. Must be in the server and currently connected to a voice channel.
- `channel` : destination voice channel — accepts an ID, name, or channel object. `nil` / `0` disconnects.

muteMember

```
{{ muteMember <target> }}
```

Server-mutes the specified member. The target must currently be in a voice channel.

- `target` : a user ID, mention, or user object.

unmuteMember

```
{{ unmuteMember <target> }}
```

Removes a server-mute from the specified member.

- `target` : a user ID, mention, or user object.

deafenMember

```
{{ deafenMember <target> }}
```

Server-deafens the specified member. The target must currently be in a voice channel.

- `target` : a user ID, mention, or user object.

undeafenMember

```
{{ undeafenMember <target> }}
```

Removes a server-deafen from the specified member.

- `target` : a user ID, mention, or user object.

getTargetPermissionsIn

```
{{ $perms := getTargetPermissionsIn <memberID> <channelID> }}
```

Returns the permissions of the specified member in the given channel as a permissions bitfield.

Example

To calculate the permission in a channel other than the current channel, for which we could just use the hasPermissions or targetHasPermissions function, we will have to use bitwise operations:

```
{{ $perms := getTargetPermissionsIn .User.ID $someChannel }}
{{ $mask := bitwiseAnd .Permissions.ManageRoles $perms }}
{{ if eq $mask .Permissions.ManageRoles }}
  You have the permissions to manage roles!
{{ else }}
  You do not have the permissions to manage roles!
{{ end }}
```

hasPermissions

```
{{ $hasPerms := hasPermissions <permission> }}
```

Returns whether the member who triggered the command has all the specified permission bits. See `.Permissions` for more information.

Example

```
{{ if hasPermissions .Permissions.Administrator }}
  You have the Administrator permission!
{{ else }}
  You do not have the Administrator permission.
{{ end }}
```

hasAnyPermissions

```
{{ $hasPerms := hasPermissions <permission> }}
```

Same as hasPermissions but only requires one of the bits set.

memberAbove

```
{{ $isAbove := memberAbove <a> <b> }}
```

Returns whether member `a` is higher than member `b` in the role hierarchy. Both `a` and `b` must be a member object.

memberAboveRole

```
{{ $isAbove := memberAboveRole <member> <role> }}
```

Returns whether the specified member is higher than the specified role in the role hierarchy. `member` must be a member object, `role` must be a role object.

onlineCount

```
{{ $count := onlineCount }}
```

Returns the count of online members on the current server, including bots.

targetHasPermissions

```
{{ $hasPerms := targetHasPermissions <memberID> <permission> }}
```

Returns whether the specified member has all the specified permission bits.

targetHasAnyPermissions

```
{{ $hasPerms := targetHasAnyPermissions <memberID> <permission> }}
```

Same as targetHasPermissions but only requires one of the bits set.

Mentions

Certain mentions are escaped by default, such that they don't ping. These functions help you actually *pinging* these special mentions.

mentionEveryone

```
{{ mentionEveryone }}
```

Mentions `@everyone` without escaping it.

mentionHere

```
{{ mentionHere }}
```

Mentions `@here` without escaping it.

mentionRole

```
{{ mentionRole <role> }}
```

Mentions `role`, which may be specified by ID, mention, name, or role object, without escaping it.

mentionRoleID

```
{{ mentionRoleID <roleID> }}
```

Mentions the given role without escaping it.

mentionRoleName

```
{{ mentionRoleName <roleName> }}
```

Mentions the role with the given name without escaping it. Searches for first case-insensitive match.

Prefer mentionRoleID, as IDs are guaranteed to be unique and do not change with role edits.

Message

addMessageReactions

```
{{ addMessageReactions <channel> <messageID> <emojis ... > }}
```

Adds reactions to a message with the given ID.

- `emojis ...` : a list of emojis to add as reactions. May also be a slice of emojis.

Default emojis are best used in the Unicode format for these purposes. Custom emojis follow a specific format in these functions. Please see the example below.

Example

```
{{ addMessageReactions nil .Message.ID "👍" "👎" "yagpdb:505114640032858114" }}
```

addReactions

```
{{ addReactions <emojis ... > }}
```

Adds reactions to the message that triggered the command.

- `emojis ...` : a list of emojis to add as reactions. May also be a slice of emojis.

addResponseReactions

```
{{ addResponseReactions <emojis ... > }}
```

Adds reactions to the response message.

- `emojis ...` : a list of emojis to add as reactions. May also be a slice of emojis.

Note that a message sent via sendMessage is not the response---use addMessageReactions for that.

complexMessageEdit

```
{{ $message := complexMessageEdit [allowed_mentions] [content] [embed] [silent] [suppress_embeds] [is_components_v2] }}
```

Creates a complex message object for use in editMessage or editMessageNoEscape.

- `allowed_mentions` : an sdict with the following keys:
  - `parse` : a slice of accepted values for mentions. May include `users` , `roles` , and `everyone` .
  - `users` : a slice of user IDs to mention.
  - `roles` : a slice of role IDs to mention.
  - `replied_user` : whether to mention the replied user.
- `content` : the new content for the message.
- `embed` : an embed object or a slice of up to 10 embed objects.
- `silent` : whether to suppress push and desktop notifications.
- `suppress_embeds` : whether to suppress embeds in the message.
- `is_components_v2` : whether the message uses components v2. Irreversibly switches the message to use components v2.

All of these keys are optional, but providing none of them will have no effect. See complexMessage for an example.

Setting `suppress_embeds` to `false` on a message with already suppressed embeds will not re-enable them.

complexMessage

```
{{ $message := complexMessage [allowed_mentions] [content] [embed] [file] [filename] [reply] [silent] [menus] [buttons]
```

Creates a complex message object for use in sendMessage functions.

- `allowed_mentions` : an sdict with the following keys:
  - `parse` : a slice of accepted values for mentions. May include `users` , `roles` , and `everyone` .
  - `users` : a slice of user IDs to mention.
  - `roles` : a slice of role IDs to mention.
  - `replied_user` : whether to mention the replied user.
- `content` : the message content.
- `embed` : an embed object or a slice of up to 10 embed objects.
- `file` : the content to print as a file.
- `filename` : the name of the file.
- `reply` : the ID of the message to reply to.
- `silent` : whether to suppress push and desktop notifications.
- `menus` : a single select menu object.
- `buttons` : a slice of button objects.
- `sticker` : single sticker ID or a slice of sticker IDs
- `forward` : an sdict containing `channel` and `message` keys specifying the message to forward
- `suppress_embeds` : whether to suppress embeds in the message.
- `is_components_v2` : whether the message uses components v2. Irreversibly switches the message to use components v2.

All of these keys are optional, but providing an empty content, file, or no embeds will result in no message being sent.

Example

The following example will output a message with an embed, some content, and a file attachment. It will also reply to the triggering message and ping the author of that message, but suppress the resulting notification.

```
{{ $message := complexMessage
  "allowed_mentions" (sdict
    "replied_user" true
  )
  "content" "Hello, world!"
  "embed" (cembed
    "title" "Embed Title"
    "description" "Embed Description"
    "color" 0xff0000
  )
  "file" "This is a file."
  "filename" "example.txt"
  "reply" .Message.ID
  "silent" true
}}
{{ sendMessage nil $message }}
```

Example for sending a sticker message:

```
{{ sendMessage nil ( complexMessage
  "sticker" 749054660769218631
) }}
```

Example for sending multiple stickers in a message:

```
{{ sendMessage nil ( complexMessage
  "sticker" ( cslice 749054660769218631 819128604311027752 1308344035563274291 )
) }}
```

Example of a message forward:

```
{{ sendMessage nil (complexMessage
    "forward" (sdict
        "channel" .Channel.ID
        "message" .Message.ID
    )
) }}
```

deleteAllMessageReactions

```
{{ deleteAllMessageReactions <channel> <messageID> [emojis ...] }}
```

Deletes all reactions from a message, optionally constrained to specific emojis.

- `emojis` : the emojis to delete. May also be a slice. Not providing this argument will delete any and all reactions.

deleteMessageReaction

```
{{ deleteMessageReaction <channel> <messageID> <userID> <emojis ...> }}
```

Deletes a specific user's reaction from a message.

- `userID` : the ID of the user whose reaction to delete.
- `emojis ...` : the emojis to delete. May also be a slice.

deleteMessage

```
{{ deleteMessage <channel> <messageID> [delay] }}
```

Deletes the specified message.

- `delay` : an optional delay in seconds to delete the message after. Defaults to 10 seconds. Max 86400 seconds (1 day).

deleteResponse

```
{{ deleteResponse [delay] }}
```

Deletes the response message.

- `delay` : an optional delay in seconds to delete after. Defaults to 10 seconds. Max 86400 seconds (1 day).

deleteTrigger

```
{{ deleteTrigger [delay] }}
```

Deletes the triggering message.

- `delay` : an optional delay in seconds to delete after. Defaults to 10 seconds. Max 86400 seconds (1 day).

editMessageNoEscape

```
{{ editMessageNoEscape <channel> <messageID> <newMessageContent> }}
```

Edits the given message without escaping mentions.

- `newMessageContent` : the new content for the message. May also be the result from complexMessageEdit.

editMessage

```
{{ editMessage <channel> <messageID> <newMessageContent> }}
```

Edits the given message with escaping mentions.

- `newMessageContent` : the new content for the message. May also be the result from complexMessageEdit.

getMessage

```
{{ $message := getMessage <channel> <messageID> }}
```

Returns the message object for the given message ID in the specified channel.

pinMessage

```
{{ pinMessage <channel> <messageID> }}
```

Pins the specified message.

publishMessage

```
{{ publishMessage <channel> <messageID> }}
```

Publishes the specified message.

publishResponse

```
{{ publishResponse }}
```

Publishes the response message.

For this to work, the custom command must be running in such an announcement channel.

sendDM

> **Note: Self-Host Addition**
>
> In this self-hosted fork (GroundPower/yagpdb-selfhost) `sendDM` is extended to optionally DM an **arbitrary user**. Upstream YAGPDB only DMs the triggering user.

```
{{ sendDM <userID> <message> }}
{{ sendDM <message> }}
```

Sends a direct message. When called with two or more arguments and the first resolves to a user (ID, mention, or user object), the direct message is sent to **that** user. When called with a single argument it falls back to the stock behaviour and DMs the triggering user.

- `userID` : *(optional, self-host)* the target user to DM — an ID, mention, or user object.
- `message` : the message to send. May also be the result from a complexMessage call.

```
{{/* DM a specific user */}}
{{ sendDM 140574551087120384 "Someone just won the game!" }}

{{/* Stock behaviour: DM the command invoker */}}
{{ sendDM "A private message just for you." }}
```

sendMessageNoEscapeRetID

```
{{ $messageID := sendMessageNoEscapeRetID <channel> <message> }}
```

Same as sendMessageNoEscape, but also returns the message ID.

sendMessageNoEscape

```
{{ sendMessageNoEscape <channel> <message> }}
```

Same as sendMessage, but does not escape mentions.

sendMessageRetID

```
{{ $messageID := sendMessageRetID <channel> <message> }}
```

Same as sendMessage, but also returns the message ID.

sendMessage

```
{{ sendMessage <channel> <message> }}
```

Sends a message in the specified channel.

- `message`: the message to send. May also be the result from a complexMessage call.

unpinMessage

```
{{ unpinMessage <channel> <messageID> }}
```

Unpins the specified message.

Regex Functions

> **Tip: Use Backticks for Regular Expressions**
>
> When supplying regular expressions to the following functions, it is convenient to use backticks (`) instead of double quotation marks. The backticks ensure that the content is interpreted verbatim as a *raw string*, which avoids needing to double-escape backslashes.
>
> For example, to search for the regular expression `\w+` in the string `$s`:
>
> ```
> {{ reFind `\w+` $s }} {{/* ok */}}
> {{ reFind "\w+" $s }} {{/* not ok; \w is interpreted as string escape sequence */}}
> {{ reFind "\\w+" $s }} {{/* also ok; same result as first line */}}
> ```

reFind

```
{{ $result := reFind <regex> <text> }}
```

Returns the first match of the regular expression `regex` in `text`, or the empty string if the pattern did not match anywhere.

reFindAll

```
{{ $result := reFindAll <regex> <text> [count] }}
```

Returns a slice of successive matches of `regex` in `text`. If `count` is provided, the number of matches is limited to that amount; otherwise, all matches are returned.

reFindAllSubmatches

```
{{ $result := reFindAllSubmatches <regex> <text> [count] }}
```

Returns a slice of successive submatches of `regex` in `text`. Each submatch is itself a slice containing the match of the entire expression, followed by any matches of capturing groups. If `count` is provided, the number of submatches is limited to that amount; otherwise, all submatches are returned.

reQuoteMeta

```
{{ $result := reQuoteMeta <string> }}
```

Escapes all regular expression metacharacters in the input `string`; the result is a regular expression matching the literal input string.

reReplace

```
{{ $result := reReplace <regex> <text> <replacement> }}
```

Replaces all matches of `regex` in `text` with `replacement`.

reSplit

```
{{ $result := reSplit <regex> <text> [count] }}
```

Splits the `text` around each match of `regex`, returning a slice of delimited substrings.

if the `count` parameter is specified, it limits the number of substrings to return:

- `count > 0`: at most `count` substrings; the last substring will be the unsplit remainder;
- `count == 0`: the result is `nil` (zero substrings);
- `count < 0`: all substrings.

Role Functions

addRole

```
{{ $role := addRole <role> [delay] }}
```

Adds the specified role to the triggering member.

- `role`: a role ID, mention, name, or role object.
- `delay`: an optional delay in seconds.

addRoleID

```
{{ addRoleID <roleID> [delay] }}
```

Adds the specified role ID to the triggering member.

- `delay`: an optional delay in seconds.

addRoleName

```
{{ addRoleName <roleName> [delay] }}
```

Adds the first case-insensitive matching role name to the triggering member.

- `delay`: an optional delay in seconds.

getRole

```
{{ $role := getRole <role> }}
```

Returns a role object. `role` may either be an ID or a name to match against (ignoring case).

getRoleID

```
{{ $role := getRoleID <roleID> }}
```

Returns a role object by its ID.

getRoleName

```
{{ $role := getRoleName <roleName> }}
```

Returns a role object by its name (case-insensitive).

giveRole

```
{{ giveRole <member> <role> [delay] }}
```

Gives the specified role to the target member.

- `role`: a role ID, mention, name, or role object.
- `delay`: an optional delay in seconds.

giveRoleID

```
{{ giveRoleID <member> <roleID> [delay] ]}}
```

Gives the role specified by its ID to the target member.

- `member`: the member to target. Either an ID, mention, or user object, but must be part of the server.
- `delay`: an optional delay in seconds.

giveRoleName

```
{{ giveRoleName <member> <roleName> [delay] }}
```

Gives the first case-insensitive matching role name to the target member.

- `delay`: an optional delay in seconds.

hasRole

```
{{ $result := hasRole <role> }}
```

Reports whether the triggering member has the specified role.

- `role`: a role ID, mention, name, or role object.

hasRoleID

```
{{ $result := hasRoleID <roleID> }}
```

Reports whether the triggering member has the specified role ID.

hasRoleName

```
{{ $result := hasRoleName <roleName> }}
```

Reports whether the triggering member has the specified role name (case-insensitive).

removeRole

```
{{ removeRole <role> [delay] }}
```

Removes the specified role from the triggering member.

- `role`: a role ID, mention, name, or role object.
- `delay`: an optional delay in seconds.

removeRoleID

```
{{ removeRoleID <roleID> [delay] }}
```

Removes the role with the specified ID from the triggering member with an optional delay in seconds.

removeRoleName

```
{{ removeRoleName <roleName> [delay] }}
```

Removes the first case-insensitive matching role name from the triggering member with an optional delay in seconds.

roleAbove

```
{{ $result := roleAbove <role1> <role2> }}
```

Reports whether `role1` is above `role2` in the role hierarchy. Both arguments must be a role object.

setRoles

```
{{ setRoles <target> <newRoles> }}
```

Sets the roles of the specified user to the provided list of role IDs. The roles are overwritten, so any existing roles are removed if not included in the new list. `newRoles` must be a slice. `target` may be a user ID, mention, or user object, but must be a member of the server.

takeRole

```
{{ takeRole <target> <role> [delay] }}
```

Removes the specified role from the target member.

- `target`: a user ID, mention, or user object. The target must be part of the server.
- `role`: a role ID, mention, name or role object.
- `delay`: an optional delay in seconds.

takeRoleID

```
{{ takeRoleID <target> <roleID> [delay] }}
```

Removes the specified role from `target`. `target` may be a user ID, mention, or user object, but must be a member of the server.

- `delay`: an optional delay in seconds.

takeRoleName

```
{{ takeRoleName <target> <roleName> [delay] }}
```

Removes the first case-insensitive matching role name from `target` with an optional delay in seconds. `target` may be a user ID, mention, or user object, but must be a member of the server.

targetHasRole

```
{{ $result := targetHasRole <target> <role> }}
```

Reports whether the target member has the specified role.

- `role`: a role ID, mention, name or role object.

targetHasRoleID

```
{{ $result := targetHasRoleID <target> <roleID> }}
```

Reports whether the specified target has the specified role ID. `target` may be a user ID, mention, or user object, but must be a member of the server.

targetHasRoleName

```
{{ $result := targetHasRoleName <target> <roleName> }}
```

Reports whether the specified target has the specified role name (case-insensitive). `target` may be a user ID, mention, or user object, but must be a member of the server.

String Manipulation

> **Note: RegEx Limitations**
> All regexp functions are limited to 10 different pattern calls per CC.

hasPrefix

```
{{ $result := hasPrefix <string> <prefix> }}
```

Reports whether the given `string` begins with `prefix`.

hasSuffix

```
{{ $result := hasSuffix <string> <suffix> }}
```

Reports whether the given `string` ends with `suffix`.

joinStr

```
{{ $result := joinStr <separator> <args ... > }}
```

Concatenates `args ...` in order, inserting the given `separator` between consecutive arguments and returns the result. As a special case, slices of strings are formatted as if each element was provided separately, so

```
{{ joinStr " " (cslice "cat" "dog") }}
```

yields `cat dog`.

See also printf if you just want to concatenate arguments without a separator.

lower

```
{{ $result := lower <string> }}
```

Converts `string` to all lowercase and returns the result.

print

```
{{ $result := print <args ... > }}
```

Concatenates the arguments in order, adding spaces between arguments when neither is a string.

println

```
{{ $result := println <args ... > }}
```

Concatenates the arguments in order, adding spaces between arguments and inserting a newline at the end.

printf

```
{{ $result := printf <format> <args ... > }}
```

Interpolates `args ...` according to `format`. See the Go `fmt` package documentation.

sanitizeText

```
{{ $result := sanitizeText <string> }}
```

Replaces accented and confusable characters in `string` with their normal, ISO-Latin variants.

split

```
{{ $result := split <string> <separator> }}
```

Splits `string` around each instance of `separator`, returning a slice of delimited substrings.

title

```
{{ $result := title <string> }}
```

Returns the string with the first letter of each word capitalized. (Title Case)

trimSpace

```
{{ $result := trimSpace <string> }}
```

Returns the string with all leading and trailing white space removed.

upper

```
{{ $result := upper <string> }}
```

Converts `string` to all uppercase and returns the result.

Time

currentTime

```
{{ $time := currentTime }}
```

Returns the current time in UTC.

formatTime

```
{{ $formatted := formatTime <time> <layout> }}
```

Formats `time` according to `layout`. Within the layout string, certain phrases represent placeholders that are replaced with the actual data from `time`: for instance, `Monday` is replaced with the weekday. A list of common placeholders follows; see the Go `time` package documentation for the full list.

Placeholder	Meaning
Mon	Weekday (abbreviated)
Monday	Weekday (full name)
2	Day of month (single digit)
02	Day of month (zero padded)
Jan	Month (abbreviated)
January	Month (full name)
1	Month (single digit)
01	Month (zero padded)
15	Hour (24-hour format)
3	Hour (12-hour format)
04	Minute (zero padded)
05	Second (zero padded)
MST	Timezone (abbreviated)
2006	Year (full year)
PM	AM-PM

humanizeDurationHours

```
{{ $formatted := humanizeDurationHours <duration> }}
```

Returns `duration` as a human-readable string, rounded down to the nearest hour.

humanizeDurationMinutes

```
{{ $formatted := humanizeDurationMinutes <duration> }}
```

Returns `duration` as a human-readable string, rounded down to the nearest minute.

humanizeDurationSeconds

```
{{ $formatted := humanizeDurationSeconds <duration> }}
```

Returns `duration` as a human-readable string, rounded down to the nearest second.

humanizeTimeSinceDays

```
{{ $formatted := humanizeTimeSinceDays <time> }}
```

Returns the duration that has passed since the specified `time` as a human-readable string, rounded down to the nearest day.

loadLocation

```
{{ $location := loadLocation "location" }}
```

Searches the IANA Time Zone database for the given location name, returning the corresponding location object on success. (Given a time object `$time`, `$time.In $location` then returns a copy of the time set in the given location for display purposes.)

As a special case, providing `UTC`, or the empty string `""`, yields the UTC location. Providing `Local` yields the local time zone of the host YAGPDB server.

newDate

```
{{ $time := newDate <year> <month> <day> <hour> <minute> <second> [location] }}
```

Returns the time object corresponding to

```
yyyy-mm-dd hh:mm:ss
```

In the appropriate zone for that time in the given location.

The month, day, hour, min, and sec values may be outside their usual ranges and will be normalized during the conversion. For example, October 32 converts to November 1.

A daylight savings time transition skips or repeats times. For example, in the United States, March 13, 2011 2:15am never occurred, while November 6, 2011 1:15am occurred twice. In such cases, the choice of time zone, and therefore the time, is not well-defined. `newDate` returns a time that is correct in one of the two zones involved in the transition, but it does not guarantee which.

parseTime

```
{{ $time := parseTime <input> <layout> [location] }}
```

Undos the operation performed by `formatTime`: that is, given some `input` string representing a time using the given `layout`, `parseTime` returns the corresponding time object in the specified location, or UTC by default. If the input is invalid or does not follow `layout`, the zero time is returned.

A slice of layouts may be provided, in which case the input is matched against each in order until one matches or the end of the slice is reached.

snowflakeToTime

```
{{ $time := snowflakeToTime <snowflake> }}
```

Returns the UTC time at which the given Discord snowflake was created.

timestampToTime

```
{{ $time := timestampToTime <unixSeconds> }}
```

Returns the UTC time corresponding to the given UNIX time, measured in seconds since January 1, 1970.

weekNumber

```
{{ $week := weekNumber <time> }}
```

Returns the ISO 8601 week number in which the time occurs, ranging between 1 and 53. Jan 01 to Jan 03 of year n might belong to week 52 or 53 of year n-1, and Dec 29 to Dec 31 might belong to week 1 of year n+1.

> **Note: Discord Timestamp Formatting**
>
> Discord Timestamp Styles referenced on Discord message documentation can be done using the `print` function:
>
> `{{print "<t:" currentTime.Unix ":F>"}}` for "Long Date/Time" formatting.

Type Conversion

structToSdict

```
{{ $sdict := structToSdict <struct> }}
```

Constructs a sdict form the fields of `struct`.

toByte

```
{{ $bytes := toByte <string> }}
```

Converts `string` to a slice of UTF-8 bytes.

toDuration

```
{{ $duration := toDuration <x> }}
```

Converts the input, which may be a number (interpreted as nanoseconds) or a duration string such as `5m`, to a duration object. Returns the zero duration for invalid inputs.

toFloat

```
{{ $y := toFloat <x> }}
```

Converts the input `x` to a float64, returning zero for invalid inputs.

toInt64

```
{{ $n := toInt64 <x> [base] }}
```

Converts the input to an int64 in the given base (0, 2 to 36), returning zero for invalid inputs. Defaults to base 10 (decimal) if not specified.

toInt

```
{{ $n := toInt <x> [base] }}
```

Converts the input to an integer in the given base (0, 2 to 36), returning zero for invalid inputs. Defaults to base 10 (decimal) if not specified.

toRune

```
{{ $runes := toRune <string> }}
```

Converts the given string to a slice of runes (Unicode code points).

toString

Aliases: `str`.

```
{{ $str := toString <x> }}
```

Converts the input to a string, returning the empty string for invalid inputs.

User

currentUserAgeHuman

```
{{ $age := currentUserAgeHuman }}
```

Returns the account age of the current user as a human-readable string.

currentUserAgeMinutes

```
{{ $age := currentUserAgeMinutes }}
```

Returns the account age of the current user in a human-readable format, rounded down to minutes.

currentUserCreated

```
{{ $time := currentUserCreated }}
```

Returns the time object corresponding to when the current user was created.

userArg

```
{{ $user := userArg <input> }}
```

Returns the full user object specified by `input`, which can be an ID or a mention.

Miscellaneous

adjective

```
{{ $adj := adjective }}
```

Returns a random adjective.

cembed

```
{{ $embed := cembed [title] [url] [description] [color] [fields] [author] [thumbnail] [image] [footer] }}
```

Returns an embed object to send with sendMessage-related functions.

All keys are optional, but the Discord API will reject completey empty embeds, so *some* content is required.

- `title` : the title of the embed
- `url` : the URL to hyperlink the title with
- `description` : the main text
- `color` : which color to display on the left side of the embed
- `fields` : a slice of sdicts with the following keys:
  - `name` : the name of the field
  - `value` : which text to have inside this field
  - `inline` : an optional boolean whether this field should be displayed in-line with other fields
- `author` : Shows some details at the very top of the embed. Is an sdict with the following keys:
  - `name` : The name of the author
  - `url` : the URL to hyperlink the name with
  - `icon_url` : the author's icon
- `thumbnail` : a small image in the top-right corner. Is an sdict with the following keys:
  - `url` : the image's URL
- `image` : an image to display at full width at the bottom of the embed. Is an sdict with the following keys:
  - `url` : the image's URL
- `footer` : Shows some details at the very bottom of the embed. Is an sdict with the following keys:
  - `text` : the footer's text
  - `icon_url` : a small icon to display to the left of the footer's text
- `timestamp` : a (static) timestamp to display to the right of the footer's text

> **Tip: Custom Commands Embed Generator**
> To help you get used to the embed structure in custom commands, check out https://yagpdbembeds.netlify.app, a community-made embed visualizer for YAGPDB's custom command system.

createTicket

```
{{ $ticket := createTicket <author> <topic> }}
```

Creates a new ticket associated to the specified author with given topic, returning a template ticket for that ticket.

- `author`: the member to associate this ticket with.
- `topic`: the topic of this ticket. Must be a string.

> **Warning: Dependency on Ticketing System**
>
> For this function to work correctly, the ticketing system must be enabled.

cslice

```
{{ $slice := cslice [values ... ] }}
```

Creates a slice of the provided values.

dict

```
{{ $dict := dict [values ... ] }}
```

Creates a dictionary from the provided key-value pairs. The number of parameters must be even.

execAdmin

```
{{ $resp := execAdmin "command" [args ... ] }}
```

Runs the given command with the provided (optional) arguments as the bot and returns the response.

This will not work for commands which have their response marked as a manual response, i.e. `wouldyourather` and `poll`.

execTemplate

```
{{ execTemplate "template" [data ... ] }}
```

Executes the associated `"template"` template, optionally with data. Please see Associated Templates.

exec

```
{{ $resp := exec "command" [args ... ] }}
```

Executes the given command with the provided (optional) arguments as the triggering user and returns the response.

This will not work for commands which have their response marked as a manual response, i.e. `wouldyourather` and `poll`.

getWarnings

```
{{ $warnings := getWarnings <user> }}
```

Returns a slice of warnings imposed on the specified user.

- `user`: the user to get the warnings for. Can be either a user ID or a user object.

humanizeThousands

```
{{ $formatted := humanizeThousands <number> }}
```

Places commas to separate groups of thousands in a number.

- `number`: the number to format. Must be an int or a string. Must be a whole number.

in

```
{{ $result := in <sequence> <value> }}
```

Returns whether the case-sensitive `value` is in the `sequence`. `sequence` can be a slice or string.

inFold

```
{{ $result := inFold <sequence> <value> }}
```

Same as in, but is case-insensitive. `sequence` can be a slice or string.

index

```
{{ $item := index <list> <index> }}
```

Returns the item at the specified index in `list`.

`list` is zero-indexed, so the first item is at index 0. `list` can be a slice, map, or string.

If you are indexing a map (that is, `dict` or `sdict`), `index` must be matching the key (as maps are unordered). Indexing a string type returns the character at that position as a rune.

You may optionally add additional indices in case you have nested structures, like `index $x 0 1`. This is equivalent to chaining `index` calls, e.g. `index (index $x 0) 1`.

kindOf

```
{{ $kind := kindOf <value> [indirect] }}
```

Returns the kind of the provided value.

If `value` is behind an `interface{}` or pointer, set `indirect` to true to read the inner value. Most users of this function will want to do this.

len

```
{{ $length := len <list> }}
```

Returns the length of the provided list. `list` can be a slice, map, or string.

noun

```
{{ $noun := noun }}
```

Returns a random noun.

parseArgs

```
{{ $args := parseArgs <requiredArgs> <errorMessage> [... cargs] }}
```

Parses arguments passed to the custom command. Ensures that at least `requiredArgs` are passed and checks that these arguments match as specified by `carg ...`, emits `errorMessage` otherwise.

Passing an empty string as `errorMessage` will generate one for you based on the provided argument definitions.

The result has the `.Get N` and `.IsSet N` methods available, returning the value or reporting whether the argument is present, respectively, at position `N` (starting from 0).

- `... cargs`: a list of argument definitions. Must have at least `requiredArgs` elements. Has the following arguments:
  - `"type"`: the type of this argument as a quoted string.
  - `"name"`: the name of this argument. Must be a string.

An argument's `"type"` must be one of the following:

- `int`: whole number
- `float`: decimal number
- `string`: text
- `user`: user mentions, resolves to a user object
- `userid`: mentions or user IDs, resolves to the ID itself
- `member`: mentions or user IDs, resolves to a member object
- `channel`: channel mention or ID, resolves to a channel object
- `role`: role name or ID, resolves to a role object
- `duration`: duration that is human-readable, i.e `10h5m` or `10 hour 5 minutes` would both resolve to the same duration

Additionally, the `int`, `float`, and `duration` type support validation ranges in the interval `(min, max)`, where for `duration` it is in time.Duration format values.

Example

```
{{ $args := parseArgs 1 "" (carg "int" "coolness level" 0 100) (carg "member" "target member") }}
Coolness: {{ $args.Get 0 }}
{{- if $args.IsSet 1 }}
Target: {{ ($args.Get 1).User }}
{{- else }}
Target: {{ .User }}
{{ end }}
```

Please see our in-depth guide in the learning resources for a full breakdown of this function and how to make good use of it.

sdict

```
{{ $map := sdict [values ... ] }}
```

Creates a dictionary from the provided key-value pairs. The number of parameters must be even. The keys must be of string type.

sendTemplateDM

```
{{ $messageID := sendTemplateDM "template" [data] }}
```

Same as sendTemplate, but sends it to the triggering user's direct messages instead and returns the *response* message's ID.

sendTemplate

```
{{ $messageID := sendTemplate <channel> "template" [data] }}
```

Sends an Associated Template to `channel`, with optional `data`, returning the *response* message's ID.

seq

```
{{ $sequence := seq <start> <stop> }}
```

Creates a new slice with integer-type elements, starting from `start` and ending at `stop-1`. `start` and `stop` must be whole numbers. Limited to 10,000 elements.

shuffle

```
{{ $shuffled := shuffle <list> }}
```

Returns a shuffled (randomized) version of the provided list.

sleep

```
{{ sleep <seconds> }}
```

Pauses the execution of the custom command for the specified number of seconds. The maximum duration is 60 seconds, combined across all `sleep` calls within the custom command and its associated templates.

slice

```
{{ $result := slice <item> <start> [end] }}
```

Returns a subslice of the input `item` (which may be an array, slice, or string) containing the elements starting at index `start` (inclusive), and ending at index `end`, exclusive. If only `start` is provided, it is interpreted as the start index and the subslice extends to the end of `item`.

sort

```
{{ $sorted := sort <list> [options] }}
```

Returns the given list in a sorted order. The list's items must all be of the same type. The optional `options` argument is an sdict with the following (optional) keys:

- `key`: if sorting a list of maps, the key's value to sort by. This key must be present on all maps in the slice.
- `reverse`: whether to sort in reverse (descending) order. Default: `false`.

Limited to 1 call on regular servers and 3 calls on premium servers.

verb

```
{{ $verb := verb }}
```

Returns a random verb.