

---

# Vex-Cogs

Vexed

Oct 23, 2021



## GETTING STARTED:

<b>1</b>	<b>Getting my cogs</b>	<b>3</b>
<b>2</b>	<b>Opt-in Telemetry and Error Reporting</b>	<b>5</b>
<b>3</b>	<b>Technical details of Sentry</b>	<b>7</b>
<b>4</b>	<b>AnotherPingCog</b>	<b>11</b>
<b>5</b>	<b>Aliases</b>	<b>15</b>
<b>6</b>	<b>Beautify</b>	<b>17</b>
<b>7</b>	<b>BetterUptime</b>	<b>19</b>
<b>8</b>	<b>CmdLog</b>	<b>21</b>
<b>9</b>	<b>GitHub</b>	<b>25</b>
<b>10</b>	<b>MadTranslate</b>	<b>29</b>
<b>11</b>	<b>StatTrack</b>	<b>31</b>
<b>12</b>	<b>Status</b>	<b>41</b>
<b>13</b>	<b>Status Reference</b>	<b>47</b>
<b>14</b>	<b>System</b>	<b>49</b>
<b>15</b>	<b>TimeChannel</b>	<b>53</b>
<b>16</b>	<b>WOL</b>	<b>55</b>
<b>17</b>	<b>Status Events</b>	<b>59</b>
<b>18</b>	<b>Changelog</b>	<b>63</b>
<b>19</b>	<b>Aliases</b>	<b>65</b>
<b>20</b>	<b>AnotherPingCog</b>	<b>67</b>
<b>21</b>	<b>Beautify</b>	<b>69</b>
<b>22</b>	<b>BetterUptime</b>	<b>71</b>

<b>23 CmdLog</b>	<b>75</b>
<b>24 GitHub</b>	<b>79</b>
<b>25 MadTranslate</b>	<b>81</b>
<b>26 StatTrack</b>	<b>83</b>
<b>27 Status</b>	<b>85</b>
<b>28 System</b>	<b>91</b>
<b>29 TimeChannel</b>	<b>95</b>
<b>30 WOL</b>	<b>97</b>
<b>31 Meta Docs</b>	<b>99</b>
<b>32 Indices and tables</b>	<b>101</b>
<b>Index</b>	<b>103</b>

Docs version: 2.1.1



## GETTING MY COGS

[p] is your prefix.

---

**Note:** You can replace `vex` with whatever you want, but you'll need to type it out every time you install one of my cogs so make sure it's simple. Note it's case sensitive.

---

1. **First, you need to add the my repository (repo):**

```
[p]repo add vex-cogs https://github.com/Vexed01/Vex-Cogs
```

2. **Now you can install my cogs with this command:**

```
[p]cog install vex-cogs cogname
```

3. **Finally, you need to load the cog:**

```
[p]load cogname
```

You can list my cogs with this command:

```
[p]cog list vex-cogs
```

---

**Tip:** It's a good idea to keep cogs up to date. You should run this command every now and again to update all your cogs:

```
[p]cog update
```

---

### 1.1 Cogs

Click a cog name to see detailed documentation.

Cog name	Summary
<i>aliases</i>	Get all the information you could ever need about a command's aliases.
<i>anotherpingcog</i>	Just another ping cog... But this one has fancy colours in an embed!
<i>beautify</i>	Beautify and minify JSON.
<i>betteruptime</i>	New uptime command that tracks the bot's uptime percentage (last 30 days).
<i>cmdlog</i>	Track command usage, searchable by user, server or command name.
<i>github</i>	Create, comment, labelify and close GitHub issues, with partial PR support.
<i>madtranslate</i>	Translate text through lots of languages. Get some funny results!
<i>stattrack</i>	Track metrics about your bot and view them in Discord.
<i>status</i>	Recieve automatic status updates from various services, including Discord.
<i>system</i>	Get system metrics of the host device, such as RAM or CPU.
<i>timechannel</i>	Get the time in different timezones in voice channels.
<i>wol</i>	Use Wake on LAN from Discord! Sends magic packets on the local network.



## OPT-IN TELEMETRY AND ERROR REPORTING

I ([github.com/Vexed01](https://github.com/Vexed01)) have **opt-in** telemetry and error reporting built into all of my ([github.com/Vexed01/Vex-Cogs](https://github.com/Vexed01/Vex-Cogs)) cogs.

Enabling or disabling it affects all of my cogs. You can view whether or not it's enabled with the `[p]vextelemetry` command. If this command doesn't exist, then no data is being sent. This is likely because you haven't updated yet.

I use a platform called Sentry ([sentry.io](https://sentry.io)) to collect this.

**No data is collected relating to command usage.**

### 2.1 Why collect this?

Error reporting allows me to fix bugs better - with more context to fix them faster and easier. Sentry has a variety of tools to help pinpoint when a bug was introduced.

Performance data is collected mainly because I can't be everywhere: I want to know if something is quite slow on some machines so I can try and optimise it.

### 2.2 What's sent?

**Where possible, only data associated with my cogs is sent.**

Everything that is sent is associated with a temporary session and permanent user UUID. This UUID is random and can't be linked directly to you or your bot. Anything that is sent includes some context/tags. This is basic information on the setup of the system to help me identify how to reproduce a bug.

For **telemetry**, the performance of background tasks and loops (for example config migration or time taken to check for updates in my Status cog) is sometimes reported. As stated in the opening of this page, no performance of commands is collected or sent.

For **error reporting**, whenever something goes wrong with my cogs (this could be a command breaking or something in a background loop) the traceback is sent along with some logs leading up to the error to try and help me work out why it happened. Sentry also sends some variables for debugging. In the future, some related config data (if applicable) might be sent. The IDs in this will be shortened to unidentifiable timestamps, as described below in *Sensitive data*

## 2.3 Sensitive data

A best effort is made to ensure that no sensitive data is sent. Client-side, all data sent is scrubbed of Discord invite links and Discord IDs are shortened to 4 digits (based on the timestamp of the ID - seconds and milliseconds) - so they can't be used to identify anything. In the very unlikely event your bot token makes it into the data, this will also be removed. For technical details, see *Data scrubbing* below. Sentry also has some data scrubbing on their side which should scrub most other sensitive fields. You can see more info on this in [their docs](#).

Data collected is sent to Sentry directly and as such I cannot see your IP address. I will never share any data from Sentry that could be used to identify anyone or stuff that gets past the filters for removing sensitive data.

## TECHNICAL DETAILS OF SENTRY

### 3.1 Data scrubbing

Data scrubbing has three parts: removal of bot token, removal of Discord invites, and shortening of Discord IDs.

A simple `str.replace()` is used to replace the bot token with `BOT-TOKEN`, if it appears for any reason.

For invites, the regex provided in [Red's utils](#) is used and invites replaced with `DISCORD-INVITE-LINK`

The shortening of Discord IDs is a little more complicated. Docs on these from Discord are [here](#) and will help explain this better. In short, the timestamp of the ID is in the ID from bytes 63 to 22. To shorten IDs, this is extracted and the seconds and milliseconds replace the ID. So, if an ID had the timestamp of `2021-08-18 19:23:45.114` the extracted data will be `5114`. This part is used because, for all intents and purposes it is random, and that it couldn't be used (on it's own) to find the full ID. This means that in the data I see on Sentry, IDs are quite likely to be unique but always the same if they occur in different places. It's sort of like hashing but worse but easier to implement with regex. This 4 digit ID is prefixed with `SHORTENED-ID-`

The exact functions can be seen at <https://github.com/Vexed01/vex-cog-utils/blob/main/vexcogutils/sentry.py>

### 3.2 How Sentry is set up, client-side

Sentry itself suggests a set-up like this:

```
import sentry_sdk

sentry_sdk.init(
    dsn=...,
    ...
)

# roughly copied from https://docs.sentry.io/platforms/python/#configure
```

However, this would **not** work if you wanted to report to multiple DSNs - something with is certainly possible if other Cog Creators use Sentry as this would override my initiation or vice versa - or even if core Red starts using Sentry again.

So, I had to go looking for a object-oriented way of using Sentry.

A Hub is created and the Client added to that. This means that the Client only takes in data when explicitly told - useful (for example) to ensure logs from other cog's aren't used as breadcrumbs.

This idea was taken from <https://github.com/getsentry/sentry-python/issues/610>

```

import sentry_sdk

# roughly copied from SentryHelper (see below)
async def get_sentry_hub(self, dsn: str, cogname: str, cogver: str) -> "Hub":
    hub = sentry_sdk.Hub(
        sentry_sdk.Client(
            dsn=dsn,
            traces_sample_rate=1.0,
            before_send=self.remove_sensitive_data,
            before_breadcrumb=self.remove_sensitive_data,
            release=f"{cogname}@{cogver}",
            debug=False,
            max_breadcrumbs=25,
        )
    )

    hub.scope.set_tag("utils_release", ...)
    hub.scope.set_tag("red_release", ...)
    hub.scope.set_user(...) # see section below called UUIDs

    hub.start_session()
    return hub

...

# there are now two ways of sending data to Sentry though that Hub:
with hub:
    sentry_sdk.add_breadcrumb(...)
# or:
hub.add_breadcrumb(...)

# for some reason you need to use the "with hub" context manager when
# capturing an exception, otherwise you can just do hub.thing() for everything else

```

### 3.2.1 SentryHelper

In Vex-Cog-Utils (VCU), as part of the client-side Sentry set up, the SentryHelper class is initiated in the `__init__.py` to the variable `sentryhelper` (`vexcogutils.sentryhelper`).

This class has various things to reduce boilerplate in each cog.

As VCU is designed to work with `importlib.reload()`, there is also an extra check to not create a new SentryHelper class if the cog is initiating from a reload (this is done through checking if `sentryhelper` is already defined as `importlib.reload()` keeps global variables).

### 3.2.2 VexTelemetry (the cog)

The SentryHelper class also adds a cog to the bot called VexTelemetry. This is what has the [p]vextelemetry command to manage whether data is sent or not. This ensures that the cog is always registered, but only once.

### 3.2.3 Config

Setup data is stored in Red's config under the fictional cog name Vex-Cog-Utills-Telemetry

### 3.2.4 Owner notifications

There are two types of messages sent to owners: "master" and "reminder":

- The "master" message is the first message to the owner when they first load one of my cogs.
- A "reminder" message will be sent whenever one of my cogs is loaded for the first time AND a master message was sent previously. If Sentry is enabled, these will be sent every time a new cog of mine is loaded. If Sentry is disabled, these will only be sent once per loading of a new cog of mine IF it is the first cog loaded since last bot restart. This has the added bonus of meaning that when this will be rolled out to all my cogs it will only send 1 DM (or at least that's the plan...)

To prevent repeated messages, a record of which cogs have been notified is stored in Config (see above)

## 3.3 How Sentry is set up, server-side

All my cogs have their own project and thus DSN. This is so they are separated.

However, they are all in the same organisation/account.

*Don't really think there's much else to put here...*

## 3.4 Only catching errors for *my* cogs

I override a function called cog\_command\_error in my cog classes. This means that all *command* errors are sent through this if they are part of this cog. To also ensure they are handled normally by Red/dpy, they are sent back to the bot's error handler with unhandled\_by\_cog=True.

```
# In the cog class
async def cog_command_error(self, ctx: "commands.Context", error: "CommandError"):
    await self.bot.on_command_error(ctx, error, unhandled_by_cog=True) # type:ignore #_
↪ Ensure main bot error handler still handles it as normal
    # Sentry logging here
```

For background loops and tasks, I generally already had full error catching and handling. I just had to send the exception to Sentry as well as log it with Python's logging module.

## 3.5 Usage of the Vex-Cog-Utills package

When I initially made VCU, it was at the back of my mind that I could one day use this for telemetry and error reporting. As such, all my cogs were already heavily integrated with these utils when I started working on adding Sentry.

## 3.6 UUIDs

I choose to use UUIDs as a way to separate users and allow for features like Release Health to work. These are generated using the standard lib uuid package:

```
import uuid  
  
uuid.uuid4() # a completely random UUID
```

## ANOTHERPINGCOG

This is the cog guide for the anotherpingcog cog. You will find detailed docs about usage and commands.

[p] is considered as your prefix.

---

**Note:** To use this cog, you will need to install and load it.

See the [Getting my cogs](#) page.

---

### 4.1 Usage

A rich embed ping command with latency timings.

You can customise the emojis, colours or force embeds with [p]pingset.

### 4.2 Commands

#### 4.2.1 ping

##### Syntax

[p]ping
---------

##### Description

A rich embed ping command with timings.

This will show the time to send a message, and the WS latency to Discord. If I can't send embeds or they are disabled here, I will send a normal message instead. The embed has more detail and is preferred.

### 4.2.2 pingset

---

**Note:** This command is locked to the bot owner.

---

#### Syntax

```
[p]pingset
```

#### Description

Manage settings - emojis, embed colour, and force embed.

#### pingset forceembed

#### Syntax

```
[p]pingset forceembed
```

#### Description

Toggle whether embeds should be forced.

If this is disabled, embeds will depend on the settings in `embedset`.

If it's enabled, embeds will always be sent unless the bot doesn't have permission to send them.

By default, this is True because the embed is richer and has more information. And it looks looks better.

This will be removed when a global per-command settings is available in Core Red.

#### pingset green

#### Syntax

```
[p]pingset green <emoji> [hex_colour=default]
```

#### Description

Set the colour and emoji to use for the colour Green.

If you want to go back to the defaults, just do `[p]pingset green default default`.

#### Arguments:

`<emoji>` Just send the emoji as you normally would. It must be a custom emoji and I must be in the sever the emoji is in. You can also put `default` to use

`[hex_colour]` (optional) The hex code you want the colour for Red to be. It looks best when this is the same colour as the emoji. Google "hex colour" if you need help with this.

#### Examples:

- `[p]pingset green :emoji: #43B581`
- `[p]pingset green :emoji: default`
- `[p]pingset green default #43B581`
- `[p]pingset green default default`



## pingset orange

### Syntax

```
[p]pingset orange <emoji> [hex_colour=default]
```

### Description

Set the colour and emoji to use for the colour Orange.

If you want to go back to the defaults, just do `[p]pingset orange default default`.

### Arguments:

`<emoji>` Just send the emoji as you normally would. It must be a custom emoji and I must be in the sever the emoji is in. You can also put `default` to use

`[hex_colour]` (optional) The hex code you want the colour for Red to be. It looks best when this is the same colour as the emoji. Google “hex colour” if you need help with this.

### Examples:

- `[p]pingset orange :emoji: #FAA61A`
- `[p]pingset orange :emoji: default`
- `[p]pingset orange default #FAA61A`
- `[p]pingset orange default default`

## pingset red

### Syntax

```
[p]pingset red <emoji> [hex_colour=default]
```

### Description

Set the colour and emoji to use for the colour Red.

If you want to go back to the defaults, just do `[p]pingset red default default`.

### Arguments:

`<emoji>` Just send the emoji as you normally would. It must be a custom emoji and I must be in the sever the emoji is in. You can also put `default` to use

`[hex_colour]` (optional) The hex code you want the colour for Red to be. It looks best when this is the same colour as the emoji. Google “hex colour” if you need help with this.

### Examples:

- `[p]pingset red :emoji: #F04747`
- `[p]pingset red :emoji: default`
- `[p]pingset red default #F04747`
- `[p]pingset red default default`

## pingset settings

### Syntax

```
[p]pingset settings
```

### Description

See your current settings.

## ALIASES

This is the cog guide for the aliases cog. You will find detailed docs about usage and commands.

[p] is considered as your prefix.

---

**Note:** To use this cog, you will need to install and load it.

See the [Getting my cogs](#) page.

---

### 5.1 Usage

Get all the alias information you could ever want about a command.

### 5.2 Commands

#### 5.2.1 aliases

##### Syntax

```
[p]aliases <command>
```

##### Description

Get all the alias information you could ever want about a command.

This will show the main command, built-in aliases, global aliases and server aliases.

##### Examples:

- [p]aliases foo
- [p]aliases foo bar



## BEAUTIFY

This is the cog guide for the beautify cog. You will find detailed docs about usage and commands.

[p] is considered as your prefix.

---

**Note:** To use this cog, you will need to install and load it.

See the [Getting my cogs](#) page.

---

### 6.1 Usage

Beautify and minify JSON.

This cog has two commands, [p]beautify and [p]minify. Both of which behave in similar ways.

They are very flexible and accept inputs in many ways, for example replies or uploading - or just simply putting it after the command.

### 6.2 Commands

#### 6.2.1 beautify

##### Syntax

```
[p]beautify [data]
```

##### Description

Beautify some JSON.

This command accepts it in a few forms.

1. Upload the JSON as a file (it can be .txt or .json) - Note that if you upload multiple files I will only scan the first one
2. Paste the JSON in the command - You send it raw, in inline code or a codeblock
3. Reply to a message with JSON - I will search for attachments and any codeblocks in the message

##### Examples:

- [p]beautify {"1": "One", "2": "Two"}
- [p]beautify (with file uploaded)
- [p]beautify (while replying to a message)

## 6.2.2 minify

### Syntax

```
[p]minify [data]
```

### Description

Minify some JSON.

This command accepts it in a few forms.

1. Upload the JSON as a file (it can be .txt or .json) - Note that if you upload multiple files I will only scan the first one  
2. Paste the JSON in the command - You send it raw, in inline code or a codeblock  
3. Reply to a message with JSON - I will search for attachments and any codeblocks in the message

### Examples:

- [p]minify {"1": "One", "2": "Two"}
- [p]minify (with file uploaded)
- [p]minify (while replying to a message)

## BETTERUPTIME

This is the cog guide for the betteruptime cog. You will find detailed docs about usage and commands.

[p] is considered as your prefix.

---

**Note:** To use this cog, you will need to install and load it.

See the [Getting my cogs](#) page.

---

### 7.1 Usage

Replaces the core `uptime` command to show the uptime percentage over the last 30 days.

The cog will need to run for a full 30 days for the full data to become available.

### 7.2 Commands

#### 7.2.1 downtime

##### Syntax

```
[p]downtime [num_days=30]
```

##### Description

Check Red downtime over the last 30 days.

The default value for `num_days` is 30. You can put 0 days for all-time data. Otherwise, it needs to be 5 or more.

##### Examples:

- `[p]uptime`
- `[p]uptime 0` (for all-time data)
- `[p]uptime 7`

## 7.2.2 uptime

### Syntax

```
[p]uptime [num_days=30]
```

### Description

Get Red's uptime percent over the last 30 days, and when I was last restarted.

The default value for num\_days is 30. You can put 0 days for all-time data. Otherwise, it needs to be 5 or more.

### Examples:

- [p]uptime
- [p]uptime 0 (for all-time data)
- [p]uptime 7



## CMDLOG

This is the cog guide for the cmdlog cog. You will find detailed docs about usage and commands.

[p] is considered as your prefix.

---

**Note:** To use this cog, you will need to install and load it.

See the [Getting my cogs](#) page.

---

## 8.1 Usage

Log command usage in a form searchable by user ID, server ID or command name.

The cog keeps an internal cache and everything is also logged to the bot's main logs under `red.vex.cmdlog`, level INFO.

## 8.2 Commands

### 8.2.1 cmdlog

---

**Note:** This command is locked to the bot owner.

---

#### Syntax

`[p]cmdlog`

---

**Tip:** Alias: cmdlogs

---

#### Description

View command logs.

Note the cache is limited to 100 000 commands, which is approximately 50MB of RAM

### cmdlog cache

#### Syntax

```
[p]cmdlog cache
```

#### Description

Show the size of the internal command cache.

### cmdlog command

#### Syntax

```
[p]cmdlog command <command>
```

#### Description

Upload all the logs that are stored for a specific command in the cache.

This does not check it is a real command, so be careful. Do not enclose it in " if there are spaces.

You can search for a group command (eg cmdlog) or a full command (eg cmdlog user). As arguments are not stored, you cannot search for them.

#### Examples:

- [p]cmdlog command ping
- [p]cmdlog command playlist
- [p]cmdlog command playlist create

### cmdlog full

#### Syntax

```
[p]cmdlog full
```

#### Description

Upload all the logs that are stored in the cache.

### cmdlog server

#### Syntax

```
[p]cmdlog server <server_id>
```

---

**Tip:** Alias: cmdlog guild

---

#### Description

Upload all the logs that are stored for for a specific server ID in the cache.

#### Example:

- [p]cmdlog server 527961662716772392

### **cmdlog user**

#### **Syntax**

```
[p]cmdlog user <user_id>
```

#### **Description**

Upload all the logs that are stored for a specific User ID in the cache.

#### **Example:**

- [p]cmdlog user 418078199982063626



This is the cog guide for the github cog. You will find detailed docs about usage and commands.

[p] is considered as your prefix.

---

**Note:** To use this cog, you will need to install and load it.

See the *Getting my cogs* page.

---

## 9.1 Usage

Create, comment, labelify and close GitHub issues.

This cog is only for bot owners. I made it for managing issues on my cog repo as a small project, but it certainly could be used for other situations where you want to manage GitHub issues from Discord.

If you would like a way to search or view issues, I highly recommend Kowlin's approved `githubcards` cog (on the repo <https://github.com/Kowlin/Sentinel>)

At present, this cannot support multiple repos.

PRs are mostly supported. You can comment on them or close them but not merge them or create them.

Get started with the `gh howtoken` command to set your GitHub token. You don't have to do this if you have already set it for a different cog, eg `githubcards`. Then set up with `gh setrepo`.

## 9.2 Commands

### 9.2.1 gh

---

**Note:** This command is locked to the bot owner.

---

#### Syntax

[p]gh
-------

---

**Tip:** Alias: `github`

---

### Description

Command to interact with this cog.  
All commands are owner only.

### gh addlabels

#### Syntax

```
[p]gh addlabels <issue>
```

---

**Tip:** Alias: gh addlabel

---

### Description

Interactive command to add labels to an issue or PR.

### gh close

#### Syntax

```
[p]gh close <issue>
```

### Description

Close an issue or PR.

### gh comment

#### Syntax

```
[p]gh comment <issue> <text>
```

### Description

Comment on an issue or PR.

### gh commentclose

#### Syntax

```
[p]gh commentclose <issue> <text>
```

### Description

Comment on, then close, an issue or PR.

---

## gh howtoken

### Syntax

```
[p]gh howtoken
```

### Description

Instructions on how to set up a token.

## gh open

### Syntax

```
[p]gh open <title>
```

### Description

Open a new issue. Does NOT reopen.

## gh removelabels

### Syntax

```
[p]gh removelabels <issue>
```

---

**Tip:** Alias: `gh removelabel`

---

### Description

Interactive command to remove labels from an issue or PR.

## gh setrepo

### Syntax

```
[p]gh setrepo <slug>
```

### Description

Set up a repo to use as a slug (USERNAME/REPO).





## MADTRANSLATE

This is the cog guide for the madtranslate cog. You will find detailed docs about usage and commands.

[p] is considered as your prefix.

---

**Note:** To use this cog, you will need to install and load it.

See the [Getting my cogs](#) page.

---

### 10.1 Usage

Translate things into lots of languages then back to English!

This will defiantly have some funny moments... Take everything with a pinch of salt!

### 10.2 Commands

#### 10.2.1 madtranslate

##### Syntax

```
[p]madtranslate [count=15] <text_to_translate>
```

---

**Tip:** Aliases: mtranslate, mtrans

---

##### Description

Translate something into lots of languages, then back to English!

##### Examples:

- [p]mtrans This is a sentence.
- [p]mtrans 25 Here's another one.

At the bottom of the output embed is a count-seed pair. You can use this with the mtransseed command to use the same language set.

## 10.2.2 mtransseed

### Syntax

```
[p]mtransseed <count_seed> <text_to_translate>
```

### Description

Use a count-seed pair to (hopefully) get reproducible results.

They may be unreproducible if Google Translate changes its translations.

The count-seed pair is obtained from the main command, `mtrans`, in the embed footer.

### Examples:

- `[p]mtrans 15-111111 This is a sentence.`
- `[p]mtrans 25-000000 Here's another one.`

## STATTRACK

This is the cog guide for the stattrack cog. You will find detailed docs about usage and commands.

[p] is considered as your prefix.

---

**Note:** To use this cog, you will need to install and load it.

See the [Getting my cogs](#) page.

---

### 11.1 Usage

Track your bot's metrics and view them in Discord. Requires no external setup, so uses Red's config. This cog will use around 150KB per day.

Commands will output as a graph. Data can also be exported with [p]stattrack export into a few different formats.

### 11.2 Commands

#### 11.2.1 stattrack

##### Syntax

```
[p]stattrack
```

##### Description

View my stats.

#### stattrack channels

##### Syntax

```
[p]stattrack channels
```

##### Description

See how many channels there are in all my guilds

**stattrack channels categories****Syntax**

```
[p]stattrack channels categories [timespan=1 day, 0:00:00]
```

**Description**

Get categories stats.

**Arguments**

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

**Examples:**

- `[p]stattrack channels categories 3w2d`
- `[p]stattrack channels categories 5d`
- `[p]stattrack channels categories all`

**stattrack channels stage****Syntax**

```
[p]stattrack channels stage [timespan=1 day, 0:00:00]
```

**Description**

Get stage channel stats.

**Arguments**

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

**Examples:**

- `[p]stattrack channels stage 3w2d`
- `[p]stattrack channels stage 5d`
- `[p]stattrack channels stage all`

**stattrack channels text****Syntax**

```
[p]stattrack channels text [timespan=1 day, 0:00:00]
```

**Description**

Get text channel stats.

**Arguments**

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

**Examples:**

- `[p]stattrack channels text 3w2d`
- `[p]stattrack channels text 5d`

- [p]stattrack channels text all

### stattrack channels total

#### Syntax

```
[p]stattrack channels total [timespan=1 day, 0:00:00]
```

#### Description

Get total channel stats.

#### Arguments

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

#### Examples:

- [p]stattrack channels total 3w2d
- [p]stattrack channels total 5d
- [p]stattrack channels total all

### stattrack channels voice

#### Syntax

```
[p]stattrack channels voice [timespan=1 day, 0:00:00]
```

#### Description

Get voice channel stats.

#### Arguments

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

#### Examples:

- [p]stattrack channels voice 3w2d
- [p]stattrack channels voice 5d
- [p]stattrack channels voice all

### stattrack commands

#### Syntax

```
[p]stattrack commands [timespan=1 day, 0:00:00]
```

#### Description

Get command usage stats.

#### Arguments

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

#### Examples:

- [p]stattrack commands 3w2d
- [p]stattrack commands 5d
- [p]stattrack commands all

### stattrack export

#### Syntax

```
[p]stattrack export
```

#### Description

Export stattrack data.

### stattrack export csv

#### Syntax

```
[p]stattrack export csv
```

#### Description

Export as CSV

### stattrack export json

#### Syntax

```
[p]stattrack export json
```

#### Description

Export as JSON with pandas orient “split”

### stattrack messages

#### Syntax

```
[p]stattrack messages [timespan=1 day, 0:00:00]
```

#### Description

Get message stats.

#### Arguments

<timespan> How long to look for, or all for all-time data. Defaults to 1 day. Must be at least 1 hour.

#### Examples:

- [p]stattrack messages 3w2d
- [p]stattrack messages 5d
- [p]stattrack messages all

---

## stattrack ping

### Syntax

```
[p]stattrack ping [timespan=1 day, 0:00:00]
```

### Description

Get my ping stats.

Get command usage stats.

### Arguments

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

### Examples:

- `[p]stattrack ping 3w2d`
- `[p]stattrack ping 5d`
- `[p]stattrack ping all`

## stattrack servers

### Syntax

```
[p]stattrack servers [timespan=1 day, 0:00:00]
```

---

**Tip:** Alias: `stattrack guilds`

---

### Description

Get server stats.

### Arguments

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

### Examples:

- `[p]stattrack servers 3w2d`
- `[p]stattrack servers 5d`
- `[p]stattrack servers all`

## stattrack status

### Syntax

```
[p]stattrack status
```

### Description

See stats about user's statuses.

### stattrack status dnd

#### Syntax

```
[p]stattrack status dnd [timespan=1 day, 0:00:00]
```

#### Description

Get dnd stats.

#### Arguments

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

#### Examples:

- `[p]stattrack status dnd 3w2d`
- `[p]stattrack status dnd 5d`
- `[p]stattrack status dnd all`

### stattrack status idle

#### Syntax

```
[p]stattrack status idle [timespan=1 day, 0:00:00]
```

#### Description

Get idle stats.

#### Arguments

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

#### Examples:

- `[p]stattrack status idle 3w2d`
- `[p]stattrack status idle 5d`
- `[p]stattrack status idle all`

### stattrack status offline

#### Syntax

```
[p]stattrack status offline [timespan=1 day, 0:00:00]
```

#### Description

Get offline stats.

#### Arguments

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

#### Examples:

- `[p]stattrack status offline 3w2d`
- `[p]stattrack status offline 5d`



- [p]stattrack status offline all

### stattrack status online

#### Syntax

```
[p]stattrack status online [timespan=1 day, 0:00:00]
```

#### Description

Get online stats.

#### Arguments

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

#### Examples:

- [p]stattrack status online 3w2d
- [p]stattrack status online 5d
- [p]stattrack status online all

### stattrack users

#### Syntax

```
[p]stattrack users
```

#### Description

See stats about user counts

### stattrack users bots

#### Syntax

```
[p]stattrack users bots [timespan=1 day, 0:00:00]
```

#### Description

Get bot user stats.

This is the count of unique bots. They are counted once, regardless of how many servers they share with me.

#### Arguments

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

#### Examples:

- [p]stattrack users bots 3w2d
- [p]stattrack users bots 5d
- [p]stattrack users bots all

### statrack users humans

#### Syntax

```
[p]statrack users humans [timespan=1 day, 0:00:00]
```

#### Description

Get human user stats.

This is the count of unique humans. They are counted once, regardless of how many servers they share with me.

#### Arguments

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

#### Examples:

- `[p]statrack users humans 3w2d`
- `[p]statrack users humans 5d`
- `[p]statrack users humans all`

### statrack users total

#### Syntax

```
[p]statrack users total [timespan=1 day, 0:00:00]
```

#### Description

Get total user stats.

This includes humans and bots and counts users/bots once per server they share with me.

#### Arguments

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

#### Examples:

- `[p]statrack users total 3w2d`
- `[p]statrack users total 5d`
- `[p]statrack users total all`

### statrack users unique

#### Syntax

```
[p]statrack users unique [timespan=1 day, 0:00:00]
```

#### Description

Get total user stats.

This includes humans and bots and counts them once, regardless of how many servers they share with me.

#### Arguments

<timespan> How long to look for, or `all` for all-time data. Defaults to 1 day. Must be at least 1 hour.

**Examples:**

- `[p]stattrack users unique 3w2d`
- `[p]stattrack users unique 5d`
- `[p]stattrack users unique all`



## STATUS

This is the cog guide for the status cog. You will find detailed docs about usage and commands.

[p] is considered as your prefix.

---

**Note:** To use this cog, you will need to install and load it.

See the [Getting my cogs](#) page.

---

## 12.1 Usage

Automatically check for status updates.

When there is one, it will send the update to all channels that have registered to receive updates from that service.

There's also the `status` command which anyone can use to check updates wherever they want.

If there's a service that you want added, contact Vexed#3211 or make an issue on the GitHub repo (or even better a PR!).

## 12.2 Commands

### 12.2.1 status

#### Syntax

```
[p]status <service>
```

#### Description

Check for the status of a variety of services, eg Discord.

#### Example:

- [p]status discord

### 12.2.2 statusset

---

**Note:** This command is locked to server admins.

---

#### Syntax

```
[p]statusset
```

#### Description

Get automatic status updates in a channel, eg Discord.

Get started with `[p]statusset preview` to see what they look like, then `[p]statusset add` to set up automatic updates.

#### statusset add

##### Syntax

```
[p]statusset add <service> [channel]
```

##### Description

Start getting status updates for the chosen service!

There is a list of services you can use in the `[p]statusset list` command.

This is an interactive command. It will ask what mode you want to use and if you want to use a webhook. You can use the `[p]statusset preview` command to see how different options look or take a look at <https://cogdocs.vexcodes.com/en/latest/cogs/statusref.html>

If you don't specify a specific channel, I will use the current channel.

#### statusset edit

##### Syntax

```
[p]statusset edit
```

##### Description

Edit services you've already set up.

#### statusset edit mode

##### Syntax

```
[p]statusset edit mode [channel] <service> <mode>
```

##### Description

Change what mode to use for status updates.

**All:** Every time the service posts an update on an incident, I will send a new message containing the previous updates as well as the new update. Best used in a fast-moving channel with other users.

**Latest:** Every time the service posts an update on an incident, I will send a new message containing only the latest update. Best used in a dedicated status channel.

**Edit:** When a new incident is created, I will sent a new message. When this incident is updated, I will then add the update to the original message. Best used in a dedicated status channel.

If you don't specify a channel, I will use the current channel.

**Examples:**

- `[p]statusset edit mode #testing discord latest`
- `[p]statusset edit mode discord edit` (for current channel)

### **statusset edit restrict**

**Syntax**

```
[p]statusset edit restrict [channel] <service> <restrict>
```

**Description**

Restrict access to the service in the status command.

Enabling this will reduce spam. Instead of sending the whole update (if there's an incident) members will instead be redirected to channels that automatically receive the status updates, that they have permission to to view.

**Examples:**

- `[p]statusset edit restrict #testing discord true`
- `[p]statusset edit restrict discord false` (for current channel)

### **statusset edit webhook**

**Syntax**

```
[p]statusset edit webhook [channel] <service> <webhook>
```

**Description**

Set whether or not to use webhooks for status updates.

Using a webhook means that the status updates will be sent with the avatar as the service's logo and the name will be `[service] Status Update`, instead of my avatar and name.

If you don't specify a channel, I will use the current channel.

**Examples:**

- `[p]statusset edit webhook #testing discord true`
- `[p]statusset edit webhook discord false` (for current channel)

### statusset list

#### Syntax

```
[p]statusset list [service]
```

---

**Tip:** Aliases: statusset show, statusset settings

---

#### Description

List that available services and ones are used in this server.

Optionally add a service at the end of the command to view detailed settings for that service.

#### Examples:

- [p]statusset list discord
- [p]statusset list

### statusset preview

#### Syntax

```
[p]statusset preview <service> <mode> <webhook>
```

#### Description

Preview what status updates will look like.

You can also see this at <https://cogdocs.vexcodes.com/en/latest/cogs/statusref.html>

#### <service>

The service you want to preview. There's a list of available services in the [p]help statusset command.

#### <mode>

**all:** Every time the service posts an update on an incident, I will send a new message containing the previous updates as well as the new update. Best used in a fast-moving channel with other users.

**latest:** Every time the service posts an update on an incident, I will send a new message containing only the latest update. Best used in a dedicated status channel.

**edit:** Naturally, edit mode can't have a preview so won't work with this command. The message content is the same as the all mode. When a new incident is created, I will send a new message. When this incident is updated, I will then add the update to the original message. Best used in a dedicated status channel.

#### <webhook>

Using a webhook means that the status updates will be sent with the avatar as the service's logo and the name will be [service] Status Update, instead of my avatar and name.

#### Examples:

- [p]statusset preview discord all true
- [p]statusset preview discord latest false



## statusset remove

### Syntax

```
[p]statusset remove <service> [channel]
```

---

**Tip:** Aliases: statusset del, statusset delete

---

### Description

Stop status updates for a specific service in this server.

If you don't specify a channel, I will use the current channel.

### Examples:

- [p]statusset remove discord #testing
- [p]statusset remove discord (for using current channel)



## STATUS REFERENCE

Below you will see previews for different modes, and webhook.

### 13.1 Modes

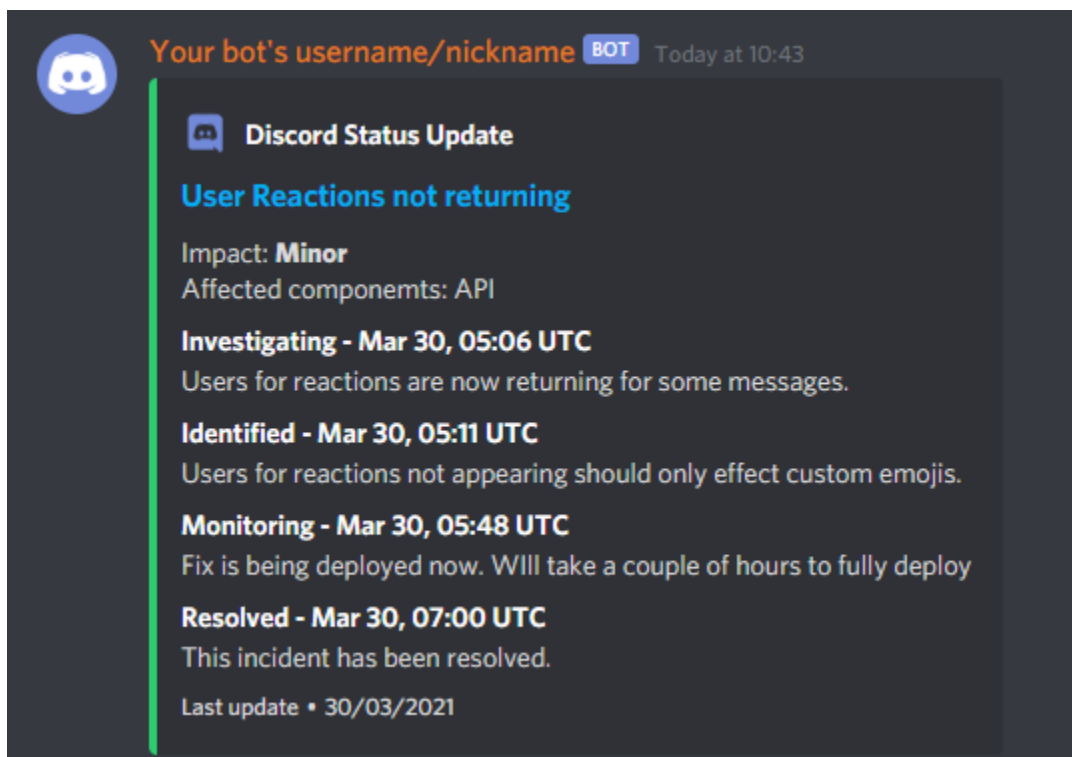
The below modes were sent *without* a webhook.

#### 13.1.1 All and Edit

---

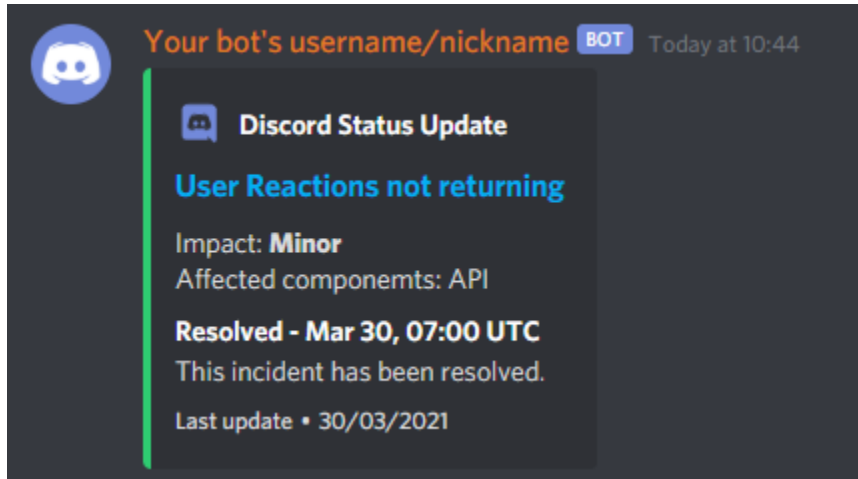
**Note:** The edit mode is the same as the all mode, however it will only send one message per incident. This message will then be edited to reflect recent updates.

---



The image shows a screenshot of a Discord chat interface. At the top left is the Discord logo. To its right, the text reads "Your bot's username/nickname BOT" in orange, followed by "Today at 10:43" in grey. Below this is a message from the bot, indicated by a speech bubble icon, titled "Discord Status Update". The main content of the message is in blue and white text: "User Reactions not returning". Below this, it states "Impact: **Minor**" and "Affected components: API". The message is divided into three sections by vertical lines: "Investigating - Mar 30, 05:06 UTC" with the text "Users for reactions are now returning for some messages.", "Identified - Mar 30, 05:11 UTC" with "Users for reactions not appearing should only effect custom emojis.", and "Monitoring - Mar 30, 05:48 UTC" with "Fix is being deployed now. Will take a couple of hours to fully deploy". The final section is "Resolved - Mar 30, 07:00 UTC" with "This incident has been resolved." At the bottom, it says "Last update • 30/03/2021".

### 13.1.2 Latest



## 13.2 Webhook

To stay brief only the latest mode is included, however the all and edit modes are also fully supported - just with a few more fields in the embed.



## SYSTEM

This is the cog guide for the system cog. You will find detailed docs about usage and commands.

[p] is considered as your prefix.

---

**Note:** To use this cog, you will need to install and load it.

See the [Getting my cogs](#) page.

---

### 14.1 Usage

Get system metrics.

Most commands work on all Oses or omit certian information. See the help for individual commands for detailed limitations.

### 14.2 Commands

#### 14.2.1 system

---

**Note:** This command is locked to the bot owner.

---

##### Syntax

[p]system
-----------

##### Description

Get information about your system metrics.

Most commands work on all Oses or omit certian information. See the help for individual commands for detailed limitations.

### system cpu

#### Syntax

```
[p]system cpu
```

#### Description

Get metrics about the CPU.

This will show the CPU usage as a percent for each core, and frequency depending on platform. It will also show the time spent idle, user and system as well as uptime.

Platforms: Windows, Linux, Mac OS .. Note:: CPU frequency is nominal and overall on Windows and Mac OS, on Linux it's current and per-core.

### system disk

#### Syntax

```
[p]system disk
```

---

**Tip:** Alias: `system df`

---

#### Description

Get information about disks connected to the system.

This will show the space used, total space, filesystem and mount point (if you're on Linux make sure it's not potentially sensitive if running the command a public space).

Platforms: Windows, Linux, Mac OS

---

**Note:** Mount point is basically useless on Windows as it's the same as the drive name, though it's still shown.

---

### system mem

#### Syntax

```
[p]system mem
```

---

**Tip:** Aliases: `system memory`, `system ram`

---

#### Description

Get information about memory usage.

This will show memory available as a percent, memory used and available as well as the total amount. Data is provided for both physical and SWAP RAM.

Platforms: Windows, Linux, Mac OS

## system network

### Syntax

```
[p]system network
```

---

**Tip:** Alias: system net

---

### Description

Get network stats. They may have overflowed and reset at some point.

Platforms: Windows, Linux, Mac OS

## system processes

### Syntax

```
[p]system processes
```

---

**Tip:** Alias: system proc

---

### Description

Get an overview of the status of currently running processes.

Platforms: Windows, Linux, Mac OS

## system sensors

### Syntax

```
[p]system sensors [fahrenheit=False]
```

---

**Tip:** Aliases: system temp, system temperature, system fan, system fans

---

### Description

Get sensor metrics.

This will return any data about temperature and fan sensors it can find. If there is no name for an individual sensor, it will use the name of the group instead.

Platforms: Linux

### system top

#### Syntax

```
[p]system top
```

---

**Tip:** Aliases: `system overview`, `system all`

---

#### Description

Get an overview of the current system metrics, similar to `top`.

This will show CPU utilisation, RAM usage and uptime as well as active processes.

Platforms: Windows, Linux, Mac OS

---

**Note:** This command appears to be very slow in Windows.

---

### system uptime

#### Syntax

```
[p]system uptime
```

---

**Tip:** Alias: `system up`

---

#### Description

Get the system boot time and how long ago it was.

Platforms: Windows, Linux, Mac OS

---

### system users

#### Syntax

```
[p]system users
```

---

#### Description

Get information about logged in users.

This will show the user name, what terminal they're logged in at, and when they logged in.

Platforms: Windows, Linux, Mac OS

---

**Note:** PID is not available on Windows. Terminal is usually `Unknown`

---



## TIMECHANNEL

This is the cog guide for the timechannel cog. You will find detailed docs about usage and commands.

[p] is considered as your prefix.

---

**Note:** To use this cog, you will need to install and load it.

See the *Getting my cogs* page.

---

### 15.1 Usage

Allocate a Discord voice channel to show the time in specific timezones. Updates every hour.

A list of timezones can be found here, though you should be able to enter any major city: [https://en.wikipedia.org/wiki/List\\_of\\_tz\\_database\\_time\\_zones#List](https://en.wikipedia.org/wiki/List_of_tz_database_time_zones#List)

There is a fuzzy search so you don't need to put the region in, only the city.

This cog will shrink down from the proper region names, for example `America/New_York` will become `New York`.

The `[p]timezones` command (runnable by anyone) will show the full location name.

### 15.2 Commands

#### 15.2.1 timechannelset

---

**Note:** This command is locked to server admins.

---

##### Syntax

```
[p]timechannelset
```

---

**Tip:** Alias: `tcset`

---

##### Description

Manage channels which will show the time for a timezone.

### timechannelset create

#### Syntax

```
[p]timechannelset create <timezone>
```

#### Description

Set up a time channel in this server.

The list of acceptable timezones is here (the “TZ database name” column): [https://en.wikipedia.org/wiki/List\\_of\\_tz\\_database\\_time\\_zones#List](https://en.wikipedia.org/wiki/List_of_tz_database_time_zones#List)

There is a fuzzy search, so you shouldn’t need to enter the region.

If you move the channel into a category, **click ‘Keep Current Permissions’ in the sync permissions dialogue.**

#### Examples:

- [p]tcset create New York
- [p]tcset create UTC
- [p]tcset create London
- [p]tcset create Europe/London

### timechannelset remove

#### Syntax

```
[p]timechannelset remove <channel>
```

#### Description

Delete and stop updating a channel.

For the <channel> argument, you can use its ID or mention (type #!channelname)

#### Example:

- [p]tcset remove #!channelname (the ! is how to mention voice channels)
- [p]tcset remove 834146070094282843

## 15.2.2 timezones

#### Syntax

```
[p]timezones
```

#### Description

See the time in all the configured timezones for this server.

## WOL

This is the cog guide for the wol cog. You will find detailed docs about usage and commands.

[p] is considered as your prefix.

---

**Note:** To use this cog, you will need to install and load it.

See the [Getting my cogs](#) page.

---

## 16.1 Usage

Send a magic packet (Wake on LAN) to a computer on the local network.

Get started by adding your computer with `[p]wolset add <friendly_name> <mac>`. Then you can wake it with `[p]wol <friendly_name>`.

For example, `[p]wolset add main_pc 11:22:33:44:55:66` then you can use `[p]wol main_pc`

## 16.2 Commands

### 16.2.1 wol

---

**Note:** This command is locked to the bot owner.

---

#### Syntax

```
[p]wol <machine>
```

#### Description

Wake a local computer.

You can set up a short name with `[p]wolset add` so you don't need to write out the MAC each time, or just send the MAC.

#### Examples:

- `[p]wol main_pc`
- `[p]wol 11:22:33:44:55:66`

## 16.2.2 wolset

---

**Note:** This command is locked to the bot owner.

---

### Syntax

```
[p]wolset
```

### Description

Manage your saved computer/MAC aliases for easy access.

### wolset add

#### Syntax

```
[p]wolset add <friendly_name> <mac>
```

#### Description

Add a machine for easy use with [p]wol.

<friendly\_name> **cannot** include spaces.

#### Examples:

- wolset add main\_pc 11:22:33:44:55:66
- wolset add main\_pc 11-22-33-44-55-66

### wolset list

#### Syntax

```
[p]wolset list
```

#### Description

See your added addresses.

This will send your MAC addresses to current channel.

### wolset remove

#### Syntax

```
[p]wolset remove <friendly_name>
```

---

**Tip:** Aliases: wolset del, wolset delete

---

#### Description

Remove a machine from my list of machines.

#### Examples:

- `wolset remove main_pc`



## STATUS EVENTS

The status cog has two events you can listen to, `on_vexed_status_update` and `on_vexed_status_channel_send`.

`status_channel_send` is fired in quick succession, especially on larger bots with lots of channels added, so you shouldn't do anything expensive. `status_channel_send` is dispatched after a successful channel send, so it won't be dispatched if the bot couldn't send for whatever reason.

`status_update` is dispatched with the channels the cog intends to send updates to when an update has been confirmed as a non-ghost update.

There is a guaranteed delay of 1 second between `status_update` and the first `status_channel_send` to allow you to perform an expensive process (or avoid repeated config calls for each dispatch) and then cache the result for the when `status_channel_send` dispatches for each channel so you get the timing correct and you can guarantee it was sent.

Though this is incredibly unlikely, the cog will cancel sending updates (and the subsequent `status_channel_send`) if it lasts longer than 4 minutes after it started that check for updates. Note multiple services' updates may be included in this time.

The events are linear. `on_status_update` guarantees the next `status_channel_send` will be the same update.

---

**Note:** If you are using this cog/event to get a parsed update to send yourself, note that `status_update` will not trigger if no channels are subscribed to the service - the cog only checks feeds that have channels subscribed to it.

---

**Tip:** For testing, the `statusdev checkfeed` (alias `statusdev cf`) command can be used. It will send the latest incident and scheduled maintenance for the service provided to the current channel, with the `force` parameter being `True`.

You can also use `statusdev forcestatus` (alias `statusdev fs`) which will send the latest incident to ALL channels in ALL servers that receive that service's incidents.

---

### 17.1 Example

```
@commands.Cog.listener()
async def on_vexed_status_update(self, **_kwargs):
    data = await self.config.all_channels() # get you data from config here
    self.data_cache = data

@commands.Cog.listener()
async def on_vexed_status_channel_send(self, *, service, channel_data, **_kwargs):
```

(continues on next page)

```

data = self.data_cache.get(channel_data.channel.id)
# then get it back here for each channel send to reduce config calls,
# esp on larger bots

if data is None:
    return

mention_ids = data["user_mentions"].get(service)
# if you registered in config as user_mentions
if mention_ids is None:
    return

mention_ids = [f"<@{id}>" for id in mention_ids]
await channel_data.channel.send(humanize_list(mention_ids))

```

## 17.2 Event Reference

**on\_vexed\_status\_update**(*update, service, channels, force*)

This event triggers before updates are sent to channels. See above for details.

### Parameters

- **update** (Update) – The main class with the update information, including what was sent and the settings it was sent with. It has subclasses in the attributes - see below *Custom Objects*.
- **service** (str) – The name of the service, in lower case. Guaranteed to be on of the keys in the file-level consts of `status.py`, though new services are being added over time so don't copy-paste and expect it to be one of them.
- **channels** (dict) – A dict with the keys as channel IDs and the values as a nested dict containing the settings for that channel.
- **force** (bool) – Whether or not the update was forced to update with `statusdev checkfeed/statusdev cf`

**on\_vexed\_status\_channel\_send**(*update, service, channel\_data, force*)

This is has similarities to the above event, mainly that it dispatches after an update was successfully sent to a specific channel. See above info at the top of this page for details.

### Parameters

- **update** (Update) – The main class with the update information, including what was sent and the settings it was sent with. It has subclasses - see below *Custom Objects*.
- **service** (str) – The name of the service, in lower case. Guaranteed to be on of the keys in the file-level consts of `status.py`, though new services are being added over time so don't copy-paste and expect it to be one of them.
- **channel\_data** (ChannelData) – The channel it was sent to and the associated settings. It has subclasses in the attributes - see below *Custom Objects*.
- **force** (bool) – Whether or not the update was forced to update with `statusdev checkfeed/statusdev cf`



## 17.3 Custom Objects

### 17.3.1 ChannelData

objects/channel.py (ignore the custom errors in this file) This object has all the settings that the update was sent with.

#### Attributes

**channel** (discord.TextChannel) – Idk, this might just be the channel the update was sent to.

**mode** (str) – The mode the update was sent as.

**webhook** (bool) – Whether or not it was sent as a webhook.

**edit\_id** (Dict[str, int]) – I cba to explain this, you don't need to know.

**embed** (bool) – Whether or not it was sent as an embed.

### 17.3.2 Update

objects/incidentdata.py This is a base object from which the two below are nested in.

#### Attributes

**incidentdata** (incidentdata) – The feed data from which the update was sent. See below.

**new\_fields** (List[UpdateField]) – A list of new fields since the service was last checked. Usually 1.

### 17.3.3 incidentdata

objects/incidentdata.py This is present in the incidentdata attribute of the Update object.

#### Attributes

**fields** (List[UpdateField]) – A list containing UpdateField objects

**title** (str) – The title of the incident

**time**: (datetime | None) - Parsed time, or None

**link** (str) – The incident link.

**actual\_time**: (datetime | None) - Parsed time, or None

**description** (str | None) – Exclusively used for when a scheduled incident is being sent

**incident\_id** (str) – The incident's unique ID

**scheduled\_for**: (datetime | None) If the incident sent was scheduled, this is when the event starts/started

#### Methods

**to\_dict()** – Get a dict of the data held in the object

**get\_update\_ids()** – Get the group IDs. These are unique and represent each update. See UpdateField for more information

### 17.3.4 UpdateField

objects/incidentdata.py This is present in the `fields` attribute of the above `incidentdata` object and the `new_fields` attribute of the `Update` object.

#### Attributes

**name** (str) – The name of the field

**value** (str) – The value of the field

**update\_id** (str) – The group ID of the field. These are unique unless the field was split up to accommodate embed limits

## CHANGELOG

I may sometimes push an update without incrementing the version. These will not be put in the changelog.

Usage of this for all version bumping updates started 21-04-08.

Date format throughout is YYYY-MM-DD

Jump links:

*aliases*

*anotherpingcog*

*beautify*

*betteruptime*

*cmdlog*

*github*

*madtranslate*

*statrack*

*status*

*system*

*timechannel*

*wol*

---

**Note:** Changelogs are automaticity generated. As such, there may sometimes be visual glitches as I do not check this.

---



## **19.1 1.0.5**

2021-08-24

- Add opt-in telemetry and error reporting

## **19.2 1.0.4**

2021-04-11

- Fix edge case to hide alias cog aliases if they are a built in command/command alias

## **19.3 1.0.3**

2021-04-08

- Fix logic for checking command
- Small internal cleanup (still more to do)



## ANOTHERPINGCOG

### 20.1 1.1.7

2021-10-04

- Fix OverflowError in edge cases (ANOTHERPINGCOG-2 on Sentry)

### 20.2 1.1.6

2021-08-24

- Add opt-in telemetry and error reporting

### 20.3 1.1.5

2021-07-18

- Allow customisation of embed footer (#35 by Obi-Wan3)

### 20.4 1.1.4

2021-05-09

- Potentially fix super edge case behaviour with command not registering





## BEAUTIFY

### 21.1 1.1.2

2021-08-24

- Add opt-in telemetry and error reporting

### 21.2 1.1.1

2021-04-24

- Internal: switch to `pyjson5.decode` instead of `pyjson5.loads`

### 21.3 1.1.0

2021-04-21

#### 21.3.1 User-facing changes

- Accept more values (True, False and None in that specific casing)

#### 21.3.2 Internal Changes

- Cache whether `pyjson5` is available instead of catching `NameError` each time
- Move more stuff to `utils` to better apply DRY

### 21.4 1.0.3

2021-04-21

- Add EUD key to `__init__.py`

## 21.5 1.0.2

2021-04-12

- Remove print statement
- Allow py codeblocks in replies (eg for beautifying an eval)

## 21.6 1.0.1

2021-04-12

- Use JSON5 to support Python dicts

## 21.7 1.0.0

2021-04-11

- Initial release

## BETTERUPTIME

### 22.1 2.0.6

2021-09-14

- Theoretically fix plotting error in certian situations

### 22.2 2.0.5

2021-08-24

- Add opt-in telemetry and error reporting

### 22.3 2.0.4

2021-08-11

- Fix edge case KeyError

### 22.4 2.0.3

2021-07-28

- Use Discord's new timestamp format

### 22.5 2.0.2

2021-06-21

- Add labels to uptime under 99.7% to graph

## 22.6 2.0.1

2021-06-21

- Require 4+ days of data for graph

## 22.7 2.0.0

2021-06-21

- Significant internal refactoring to make it more maintainable
- New command: `uptimegraph` - see uptime in graph form
- New command: `uptimeexport` (bot owner only) - export uptime data to CSV
- Fix removing wrong command on cog unload

## 22.8 1.6.0

2021-06-06

- Add `resetbu` command to reset all uptime data

## 22.9 1.6.0

2021-05-28

- Fix commands
- Fix config migration

## 22.10 1.5.2

2021-05-25

- Remove custom uptime command. .. There's some broken shit that I can't fix, rewrite was already planned and this will be fixed then (#23 on GitHub)

## 22.11 1.5.1

2021-05-23

- Fix deprecation warning

## 22.12 1.5.0

2021-05-23

- Move to storing and internally cache data as a Pandas Series

## 22.13 1.4.1

2021-05-09

- Fix unreachable code

## 22.14 1.4.0

2021-05-01

- Utilise an Abstract Base Class and move to VexLoop

## 22.15 1.3.0

2021-04-25

- Allow a custom timeframe in `uptime` and `downtime`, eg `uptime 7`
- Pagify the `downtime` command

## 22.16 1.2.2

- Slight logic changes for banding in `downtime` command



### 23.1 1.4.3

2021-09-05

- Guard dislash.py with TYPE\_CHECKING

### 23.2 1.4.2

2021-09-05

- Add support for dislash.py application commands

### 23.3 1.4.1

2021-08-28

- Fix AttributeError in sending com log to channel
- Fix AttributeError in handling slash commands from Kowlin's SlashInjector
- Ensure bot has send message permissions when setting log channel
- Fixes CMDLOG-2 and CMDLOG-3 on Sentry

### 23.4 1.4.0

2021-08-27

- Add new command ([p]cmdlog channel) to log commands to a channel

## 23.5 1.3.1

2021-08-24

- Add opt-in telemetry and error reporting

## 23.6 1.3.0

2021-08-12

- Support Application Commands (Slash, Message, User), both with slashinjector/dpy 1 and dpy 2

## 23.7 1.2.1

2021-08-07

- Initial discord.py 2.0 compatibility

## 23.8 1.3.0

2021-06-23

- Add content logging, by default turned off (see command `[p]cmdlog content`)
- Simplify EUD statement
- Add info on how long since cog load (how long current cache lasts) on log commands

## 23.9 1.1.0

2021-05-10

- Log command invoke message IDs
- Round cache size to 1 decimal place

## 23.10 1.0.2

2021-04-22

- Return correct size... I really thought I already did this.



## **23.11 1.0.1**

2021-04-18

- New command to view cache size (cmdlog cache)

## **23.12 1.0.0**

2021-04-18

- Initial release



## 24.1 1.0.1

2021-08-24

- Add opt-in telemetry and error reporting

*No updates since changelogs started*

Note: This cog is scheduled for deprecation in favour of a new cog *ghissues* which supports buttons, for when they are officially supported in Red



## MADTRANSLATE

### 25.1 1.0.2

2021-08-24

- Add opt-in telemetry and error reporting

### 25.2 1.0.1

2021-06-07

- Add Vex-Cog-Utills stuff

### 25.3 1.0.0

2021-06-07

- Initial release



## **26.1 1.3.2**

2021-09-14

- Fix TypeError in log for when loop overruns

## **26.2 1.3.1**

2021-08-24

- Add opt-in telemetry and error reporting

## **26.3 1.3.0**

2021-08-11

- Move to SQLite driver in Vex-Cog-Utils

## **26.4 1.1.0**

2021-06-25

- Move to SQLite for data storage for superior speed

## **26.5 1.0.1**

2021-06-12

- Count time to save to config seperatleu

## 26.6 1.0.0

2021-06-02

- Initial release



## STATUS

### 27.1 2.4.1

2021-09-14

- Limit embed value length in status command, for affected components. This did NOT affect the background loop and automatic sending of updates

### 27.2 2.4.0

2021-08-26

- Cache status updates, and therefore decrease the cooldown on the *status* command

### 27.3 2.3.12

2021-08-24

- Add opt-in telemetry and error reporting

### 27.4 2.3.11

2021-08-16

- Change service base image URL to static.vexcodes.com

### 27.5 2.3.10

2021-08-07

- Initial discord.py 2.0 compatibility

## 27.6 2.3.9

2021-06-27

- Improve embed limit handling

## 27.7 2.3.8

2021-06-22

- Move icons to GH Pages
- Make field name a zero width space for when embed fields are split

## 27.8 2.3.7

2021-06-17

- Fix edge case KeyError with service restrictions

## 27.9 2.3.6

2021-06-08

- New service - Fastly
- Handle embed description limits

## 27.10 2.3.5

2021-05-22

- Update to use Discord's new logo

## 27.11 2.3.4

2021-05-19

- Fix KeyError which could occur in edge cases

## 27.12 2.3.3

2021-05-16

- Change the colour for `investigating` to orange (from red)

## 27.13 2.3.2

2021-05-08

- Dynamic help for available services in all commands that previously had them listed

## 27.14 2.3.0

2021-05-05

- Use dedicated library (`markdownify`) for handling HTML to markdown
- Remove `pytz` for requirements and remove from code.

## 27.15 2.2.0

2021-05-01

- Use the ABC in the loop and move to `VexLoop`

## 27.16 2.1.5

2021-05-01

- Properly handle errors relating to service restrictions when removing a feed
- Improve error handling/logging in update loop
- Limit number of updates sent per service per check to 3 (eg when cog has been unloaded for a while)

## 27.17 2.1.4

2021-04-23

- Show status of components in command `status`

## 27.18 2.1.3

2021-04-22

- Use deque for cooldown

## 27.19 2.1.2

- Handle EUD data deletion requests (return None)

## 27.20 2.1.1

2021-13-04

- Minor refactoring

## 27.21 2.1.0

2021-13-04

### 27.21.1 User-facing changes

- Handle HTML tags for Oracle Cloud

### 27.21.2 Internal changes

- Utilise an Abstract Base Class
- Add some internal docstrings

## 27.22 2.0.0, 2.0.1

(backdated)

### 27.22.1 Important

**If the cog fails to load after updating** then you'll need to do the following.

---

**Note:** If you originally added my repo and didn't name it vex, replace vex with what you called it throughout.

---

1. **Uninstall status and remove my repo**

```
cog uninstall status
```

```
repo remove vex
```

## 2. Add my repo back and reinstall status

```
repo add vex https://github.com/Vexed01/Vex-Cogs
```

```
cog install vex status
```

## 3. Restart

```
restart
```

---

**Note:** If you haven't configured anything to catch the restart, you'll need to start your bot up again.

---

You should now be able to load the cog.

## 27.22.2 User-facing changes

- **BREAKING CHANGES:** Removed AWS, GCP, Twitter and Status.io. These will be automatically removed when you update.
- Added the docs page [Status Reference](#) to see previews for different modes/webhook
- All updates will now included the impact and affected components (see an example at [Status Reference](#))
- New service: GeForce NOW ([geforcenow](#))

## 27.22.3 Event Changes for developers

I highly recommend you read the docs page again at the [Status Events](#) page.

There have been significant changes to both the events.

## 27.22.4 Internal changes

- Significant re-factoring into more files and folders
- Rewrite of update checking and sending logic
- Implementation of Status API instead of parsing RSS
- Changes to how incidents are stored including config wrapper
- No longer write ETags to config (just cache)



## 28.1 1.3.9

2021-08-24

- Add opt-in telemetry and error reporting

## 28.2 1.3.8

2021-08-11

- Use correct timezone for system uptime

## 28.3 1.3.7

2021-08-09

- Fix error on d.py 2

## 28.4 1.3.6

2021-08-07

- Initial discord.py 2.0 compatibility

## 28.5 1.3.5

2021-06-30

- Change formatting of `system red` and it's corresponding section of `system all`

## 28.6 1.3.4

2021-06-29

- Fix `system all` non-embed output

## 28.7 1.3.5

2021-06-27

- Show Red's resource usage in the `system all` command
- Trigger typing for `system red` command
- Use the bot's name for Red's resource usage instead of just "Red"

## 28.8 1.3.2

2021-06-25

- Correctly display SWAP usage

## 28.9 1.3.1

2021-06-25

- New command: `[p]system red`

## 28.10 1.2.7

2021-06-18

- Make the cog compatible with WSL

## 28.11 1.2.6

2021-06-18

- Use UTC for bot uptime



## 28.12 1.2.5

2021-06-18

- Handle no CPU frequency data being available

## 28.13 1.2.4

2021-06-13

- Fix formatting of cpu

## 28.14 1.2.3

2021-06-12

- Add bot uptime to footer

## 28.15 1.2.2

2021-06-12

- Show uptime in footer for all commands
- Make embed formatting to two columns dynamic

## 28.16 1.2.1

2021-05-30

- Handle embed limits

## 28.17 1.2.0

2021-05-30

- Add command `system net`
- Use AsyncIter for the process generator

## 28.18 1.1.2

2021-05-08

- Dynamic help showing if commands are available on your system

## 28.19 1.1.1

2021-04-09

- Add missing docstring for `system uptime`
- (internal) Add stubs for `psutil`

## 28.20 1.1.0

2021-04-08

- **New command: `system uptime`**
  - shows what time the system was booted and how long ago that was
- Internal refactor, splitting commands and `psutil` parsers into two files

## 29.1 1.2.2

2021-08-24

- Add opt-in telemetry and error reporting

## 29.2 1.2.1

2021-08-07

- Initial discord.py 2.0 compatibility

## 29.3 1.2.0

2021-06-25

- You can now choose your own format. Take a look at `[p]tcset create` for some information on how to do so. You'll have to remove old channels with `[p]tcset remove`

## 29.4 1.1.1

2021-06-07

- Fix inconsistencies

## 29.5 1.1.0

2021-05-02

- Improve fuzzy timezone search

## 29.6 1.0.0

2021-05-01

- Initial release

### **30.1 1.0.5**

2021-08-24

- Add opt-in telemetry and error reporting

### **30.2 1.0.4**

2021-08-20

- More release testing...

### **30.3 1.0.3**

2021-08-20

- Still testing release workflow...

### **30.4 1.0.2**

2021-08-20

- Still testing release workflow...

### **30.5 1.0.1**

2021-08-20

- Testing release workflow, please ignore

## 30.6 1.0.0

2021-05-31

- Initial release

## 31.1 2.2.0

2021-06-21

- Directly link to each section at the top of changelog

## 31.2 2.1.1

2021-04-11

- Change intro at top to link to *Getting my cogs* instead of saying to load the cog
- Bring docs up to date with docstring in all cogs

## 31.3 2.1.0

2021-04-08

- Start versioning docs
- Fully use changelog

## 31.4 2.0.0

(backdated)

- Switch to furo theme





## INDICES AND TABLES

- genindex
- modindex
- search



## INDEX

### B

built-in function

    on\_vexed\_status\_channel\_send(), 60

    on\_vexed\_status\_update(), 60

### O

on\_vexed\_status\_channel\_send()

    built-in function, 60

on\_vexed\_status\_update()

    built-in function, 60