Red - Discord Bot Documentation

Release 3.1.5

Cog Creators

Aug 18, 2019

INSTALLATION GUIDES:

CHAPTER

ONE

INSTALLING RED ON WINDOWS

1.1 Needed Software

The following software dependencies can all be installed quickly and easily through powershell, using a trusted package manager for windows called Chocolatey

We also provide instructions for manually installing all of the dependencies.

1.1.1 Installing using powershell and chocolatey

To install via powershell, search "powershell" in the windows start menu, right-click on it and then click "Run as administrator"

Then run each of the following commands:

```
Set-ExecutionPolicy Bypass -Scope Process -Force
iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.
↪ps1'))
choco install git --params "/GitOnlyOnPath /WindowsTerminal" -y
choco install jre8 python -y; exit
```

1.1.2 Manually installing dependencies

- Python - Red needs Python 3.7.0 or greater

Note: Please make sure that the box to add Python to PATH is CHECKED, otherwise you may run into issues when trying to run Red.

- Git

Attention: Please choose the option to "Run Git from the Windows Command Prompt" in Git's setup.

- Java - needed for Audio

Attention: Please choose the "Windows Online" installer.

1.2 Installing Red

Attention: You may need to restart your computer after installing dependencies for the PATH changes to take effect.

1. Open a command prompt (open Start, search for "command prompt", then click it)
2. Create and activate a virtual environment (strongly recommended), see the section *Using venv*
3. Run **one** of the following commands, depending on what extras you want installed

---

Note: If you're not inside an activated virtual environment, include the `--user` flag with all `pip` commands.

- No MongoDB support:

```
python -m pip install -U Red-DiscordBot
```

- With MongoDB support:

```
python -m pip install -U Red-DiscordBot[mongo]
```

Note: To install the development version, replace `Red-DiscordBot` in the above commands with the following link:

```
git+https://github.com/Cog-Creators/Red-DiscordBot@V3/develop#egg=Red-
↪DiscordBot
```

1.3 Setting Up and Running Red

After installation, set up your instance with the following command:

```
redbot-setup
```

This will set the location where data will be stored, as well as your storage backend and the name of the instance (which will be used for running the bot).

Once done setting up the instance, run the following command to run Red:

```
redbot <your instance name>
```

It will walk through the initial setup, asking for your token and a prefix. You can find out how to obtain a token with this guide, section "Creating a Bot Account".

You may also run Red via the launcher, which allows you to restart the bot from discord, and enable auto-restart. You may also update the bot from the launcher menu. Use the following command to run the launcher:

```
redbot-launcher
```

CHAPTER

TWO

INSTALLING RED ON LINUX OR MAC

Warning: For safety reasons, DO NOT install Red with a root user. If you are unsure how to create a new user, see the man page for the `useradd` command.

2.1 Installing the pre-requirements

Please install the pre-requirements using the commands listed for your operating system.

The pre-requirements are:

- Python 3.7.0 or greater
- pip 9.0 or greater
- git
- Java Runtime Environment 8 or later (for audio support)

2.1.1 Arch Linux

```
sudo pacman -Syu python-pip git base-devel jre8-openjdk
```

2.1.2 CentOS 7, Fedora, and RHEL

```
yum -y groupinstall development
yum -y install https://centos7.iuscommunity.org/ius-release.rpm
sudo yum install zlib-devel bzip2 bzip2-devel readline-devel sqlite sqlite-devel \
openssl-devel xz xz-devel libffi-devel git2u java-1.8.0-openjdk
```

Complete the rest of the installation by *installing Python 3.7 with pyenv*.

2.1.3 Debian and Raspbian Stretch

Warning: Audio will not work on Raspberry Pi's **below** 2B. This is a CPU problem and *cannot* be fixed.

We recommend installing pyenv as a method of installing non-native versions of python on Debian/Raspbian Stretch. This guide will tell you how. First, run the following commands:

```
sudo apt install -y make build-essential libssl-dev zlib1g-dev libbz2-dev \
libreadline-dev libsqlite3-dev wget curl llvm libncurses5-dev libncursesw5-dev \
xz-utils tk-dev libffi-dev liblzma-dev python3-openssl git unzip default-jre
```

Complete the rest of the installation by *installing Python 3.7 with pyenv*.

2.1.4 Mac

Install Brew: in Finder or Spotlight, search for and open *Terminal*. In the terminal, paste the following, then press Enter:

```
/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/
↪master/install)"
```

After the installation, install the required packages by pasting the commands and pressing enter, one-by-one:

```
brew install python --with-brewed-openssl
brew install git
brew tap caskroom/versions
brew cask install homebrew/cask-versions/adoptopenjdk8
```

It's possible you will have network issues. If so, go in your Applications folder, inside it, go in the Python 3.7 folder then double click `Install certificates.command`

2.1.5 Ubuntu 18.04 Bionic Beaver and 18.10 Cosmic Cuttlefish

```
sudo apt install python3.7 python3.7-dev python3.7-venv python3-pip build-essential \
libssl-dev libffi-dev git unzip default-jre -y
```

2.1.6 Ubuntu 16.04 Xenial Xerus

We recommend adding the `deadsnakes` apt repository to install Python 3.7 or greater:

```
sudo apt install software-properties-common
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt update
```

Now, install python, pip, git and java with the following commands:

```
sudo apt install python3.7 python3.7-dev build-essential libssl-dev libffi-dev git \
unzip default-jre curl -y
curl https://bootstrap.pypa.io/get-pip.py | sudo python3.7
```

2.1.7 Installing Python with pyenv

On distributions where Python 3.7 needs to be compiled from source, we recommend the use of pyenv. This simplifies the compilation process and has the added bonus of simplifying setting up Red in a virtual environment.

```
curl -L https://github.com/pyenv/pyenv-installer/raw/master/bin/pyenv-installer | bash
```

After this command, you may see a warning about 'pyenv' not being in the load path. Follow the instructions given to fix that, then close and reopen your shell.

Then run the following command:

```
CONFIGURE_OPTS=--enable-optimizations pyenv install 3.7.2 -v
```

This may take a long time to complete, depending on your hardware. For some machines (such as Raspberry Pis and micro-tier VPSes), it may take over an hour; in this case, you may wish to remove the `CONFIGURE_OPTS=--enable-optimizations` part from the front of the command, which will drastically reduce the install time. However, be aware that this will make Python run about 10% slower.

After that is finished, run:

```
pyenv global 3.7.2
```

Pyenv is now installed and your system should be configured to run Python 3.7.

2.2 Creating a Virtual Environment

We **strongly** recommend installing Red into a virtual environment. See the section *Installing Red in a Virtual Environment*.

2.3 Installing Red

Choose one of the following commands to install Red.

Note: If you're not inside an activated virtual environment, include the `--user` flag with all `python3.7 -m pip` commands.

To install without MongoDB support:

```
python3.7 -m pip install -U Red-DiscordBot
```

Or, to install with MongoDB support:

```
python3.7 -m pip install -U Red-DiscordBot[mongo]
```

Note: To install the development version, replace `Red-DiscordBot` in the above commands with the following link:

```
git+https://github.com/Cog-Creators/Red-DiscordBot@V3/develop#egg=Red-DiscordBot
```

2.4 Setting Up and Running Red

After installation, set up your instance with the following command:

```
redbot-setup
```

This will set the location where data will be stored, as well as your storage backend and the name of the instance (which will be used for running the bot).

Once done setting up the instance, run the following command to run Red:

```
redbot <your instance name>
```

It will walk through the initial setup, asking for your token and a prefix. You can find out how to obtain a token with this guide, section "Creating a Bot Account".

You may also run Red via the launcher, which allows you to restart the bot from discord, and enable auto-restart. You may also update the bot from the launcher menu. Use the following command to run the launcher:

```
redbot-launcher
```

CHAPTER

THREE

INSTALLING RED IN A VIRTUAL ENVIRONMENT

Virtual environments allow you to isolate red's library dependencies, cog dependencies and python binaries from the rest of your system. It is strongly recommended you use this if you use python for more than just Red.

3.1 Using `venv`

This is the quickest way to get your virtual environment up and running, as `venv` is shipped with python.

First, choose a directory where you would like to create your virtual environment. It's a good idea to keep it in a location which is easy to type out the path to. From now, we'll call it `path/to/venv/` (or `path\to\venv\` on Windows).

3.1.1 `venv` on Linux or Mac

Create your virtual environment with the following command:

```
python3.7 -m venv path/to/venv/
```

And activate it with the following command:

```
source path/to/venv/bin/activate
```

Important: You must activate the virtual environment with the above command every time you open a new shell to run, install or update Red.

Continue reading *below*.

3.1.2 `venv` on Windows

Create your virtual environment with the following command:

```
python -m venv path\to\venv\
```

And activate it with the following command:

```
path\to\venv\Scripts\activate.bat
```

Important: You must activate the virtual environment with the above command every time you open a new Command Prompt to run, install or update Red.

Continue reading *below*.

3.2 Using `pyenv virtualenv`

Note: This is for non-Windows users only.

Using `pyenv virtualenv` saves you the headache of remembering where you installed your virtual environments. If you haven't already, install pyenv with pyenv-installer.

First, ensure your pyenv interpreter is set to python 3.7.0 or greater with the following command:

```
pyenv version
```

Now, create a virtual environment with the following command:

```
pyenv virtualenv <name>
```

Replace `<name>` with whatever you like. If you forget what you named it, use the command `pyenv versions`.

Now activate your virtualenv with the following command:

```
pyenv shell <name>
```

Important: You must activate the virtual environment with the above command every time you open a new shell to run, install or update Red.

Continue reading *below*.

Once activated, your `PATH` environment variable will be modified to use the virtual environment's python executables, as well as other executables like `pip`.

From here, install Red using the commands listed on your installation guide (*Windows* or *Non-Windows*).

Note: The alternative to activating the virtual environment each time you open a new shell is to provide the full path to the executable. This will automatically use the virtual environment's python interpreter and installed libraries.

3.3 Virtual Environments with Multiple Instances

If you are running multiple instances of Red on the same machine, you have the option of either using the same virtual environment for all of them, or creating separate ones.

Note: This only applies for multiple instances of V3. If you are running a V2 instance as well, You **must** use separate virtual environments.

The advantages of using a *single* virtual environment for all of your V3 instances are:

- When updating Red, you will only need to update it once for all instances (however you will still need to restart all instances for the changes to take effect)
- It will save space on your hard drive

On the other hand, you may wish to update each of your instances individually.

Important: Windows users with multiple instances should create *separate* virtual environments, as updating multiple running instances at once is likely to cause errors.

CHAPTER

FOUR

SETTING UP AUTO-RESTART USING SYSTEMD ON LINUX

4.1 Creating the service file

Create the new service file:

```
sudo -e /etc/systemd/system/red@.service
```

Paste the following and replace all instances of `username` with the username your bot is running under (hopefully not root):

```
[Unit]
Description=%I redbot
After=multi-user.target

[Service]
ExecStart=/home/username/.local/bin/redbot %I --no-prompt
User=username
Group=username
Type=idle
Restart=always
RestartSec=15
RestartPreventExitStatus=0
TimeoutStopSec=10

[Install]
WantedBy=multi-user.target
```

Save and exit `ctrl + O; enter; ctrl + x`

4.2 Starting and enabling the service

Note: This same file can be used to start as many instances of the bot as you wish, without creating more service files, just start and enable more services and add any bot instance name after the @

To start the bot, run the service and add the instance name after the @:

```
sudo systemctl start red@instancename
```

To set the bot to start on boot, you must enable the service, again adding the instance name after the @:

```
sudo systemctl enable red@instancename
```

If you need to shutdown the bot, you can use the `[p]shutdown` command or type the following command in the terminal, still by adding the instance name after the @:

```
sudo systemctl stop red@instancename
```

> **Warning:** If the service doesn't stop in the next 10 seconds, the process is killed. Check your logs to know the cause of the error that prevents the shutdown.

To view Red's log, you can acccess through journalctl:

```
sudo journalctl -u red@instancename
```

CHAPTER

FIVE

CUSTOMCOMMANDS COG REFERENCE

5.1 How it works

CustomCommands allows you to create simple commands for your bot without requiring you to code your own cog for Red.

If the command you attempt to create shares a name with an already loaded command, you cannot overwrite it with this cog.

5.2 Cooldowns

You can set cooldowns for your custom commands. If a command is on cooldown, it will not be triggered.

You can set cooldowns per member or per channel, or set a cooldown guild-wide. You can also set multiple types of cooldown on a single custom command. All cooldowns must pass before the command will trigger.

5.3 Context Parameters

You can enhance your custom command's response by leaving spaces for the bot to substitute.

Argument	Substitute
{message}	The message the bot is responding to.
{author}	The user who called the command.
{channel}	The channel the command was called in.
{server}	The server the command was called in.
{guild}	Same as with {server}.

You can further refine the response with dot notation. For example, {author.mention} will mention the user who called the command.

5.4 Command Parameters

You can further enhance your custom command's response by leaving spaces for the user to substitute.

To do this, simply put {#} in the response, replacing # with any number starting with 0. Each number will be replaced with what the user gave the command, in order.

You can refine the response with colon notation. For example, {0:Member} will accept members of the server, and {0:int} will accept a number. If no colon notation is provided, the argument will be returned unchanged.

Argument	Substitute
{#:Member}	A member of your server.
{#:TextChannel}	A text channel in your server.
{#:Role}	A role in your server.
{#:int}	A whole number.
{#:float}	A decimal number.
{#:bool}	True or False.

You can specify more than the above with colon notation, but those are the most common.

As with context parameters, you can use dot notation to further refine the response. For example, {0.mention:Member} will mention the Member specified.

5.5 Example commands

Showing your own avatar

```
[p]customcom add simple avatar {author.avatar_url}
[p]avatar
    https://cdn.discordapp.com/avatars/133801473317404673/
↪be4c4a4fe47cb3e74c31a0504e7a295e.webp?size=1024
```

Repeating the user

```
[p]customcom add simple say {0}
[p]say Pete and Repeat
    Pete and Repeat
```

Greeting the specified member

```
[p]customcom add simple greet Hello, {0.mention:Member}!
[p]greet Twentysix
    Hello, @Twentysix!
```

Comparing two text channel's categories

```
[p]customcom add simple comparecategory {0.category:TextChannel}  |  {1.
↪category:TextChannel}
[p]comparecategory #support #general
    Red  |  Community
```

CHAPTER

SIX

DOWNLOADER COG REFERENCE

class redbot.cogs.downloader.downloader.**Downloader**(*bot*)

Bases: redbot.core.commands.commands.Cog

await cog_install_path()

Get the current cog install path.

Returns The default cog install path.

Return type pathlib.Path

cog_name_from_instance(*instance*) → str

Determines the cog name that Downloader knows from the cog instance.

Probably.

Parameters **instance** (*object*) – The cog instance.

Returns The name of the cog according to Downloader..

Return type str

format_findcog_info(*command_name*, *cog_installable* = *None*) → str

Format a cog's info for output to discord.

Parameters

- **command_name** (*str*) – Name of the command which belongs to the cog.
- **cog_installable** (*Installable* or object) – Can be an *Installable* instance or a Cog instance.

Returns A formatted message for the user.

Return type str

await installed_cogs() → Tuple[redbot.cogs.downloader.installable.Installable]

Get info on installed cogs.

Returns All installed cogs / shared lib directories.

Return type tuple of *Installable*

await is_installed(*cog_name*) → Union[Tuple[bool, redbot.cogs.downloader.installable.Installable], Tuple[bool, None]]

Check to see if a cog has been installed through Downloader.

Parameters **cog_name** (*str*) – The name of the cog to check for.

Returns (True, Installable) if the cog is installed, else (False, None).

Return type tuple of (bool, *Installable*)

CHAPTER

SEVEN

PERMISSIONS COG REFERENCE

7.1 How it works

When loaded, the permissions cog will allow you to define extra custom rules for who can use a command.

If no applicable rules are found, the command will behave normally.

Rules can also be added to cogs, which will affect all commands from that cog. The cog name can be found from the help menu.

7.2 Rule priority

Rules set for subcommands will take precedence over rules set for the parent commands, which lastly take precedence over rules set for the cog. So for example, if a user is denied the Core cog, but allowed the `[p]set token` command, the user will not be able to use any command in the Core cog except for `[p]set token`.

In terms of scope, global rules will be checked first, then server rules.

For each of those, the first rule pertaining to one of the following models will be used:

1. User
2. Voice channel
3. Text channel
4. Channel category
5. Roles, highest to lowest
6. Server (can only be in global rules)
7. Default rules

In private messages, only global rules about a user will be checked.

7.3 Setting Rules From a File

The permissions cog can also set, display or update rules with a YAML file with the `[p]permissions yaml` command. Models must be represented by ID. Rules must be `true` for allow, or `false` for deny. Here is an example:

```
COG:
  Admin:
    78631113035100160: true
    96733288462286848: false
  Audio:
    133049272517001216: true
    default: false
COMMAND:
  cleanup bot:
    78631113035100160: true
    default: false
  ping:
    96733288462286848: false
    default: true
```

7.4 Example configurations

Locking the `[p]play` command to approved server(s) as a bot owner:

```
[p]permissions setdefaultglobalrule deny play
[p]permissions addglobalrule allow play [server ID or name]
```

Locking the `[p]play` command to specific voice channel(s) as a serverowner or admin:

```
[p]permissions setdefaultserverrule deny play
[p]permissions setdefaultserverrule deny "playlist start"
[p]permissions addserverrule allow play [voice channel ID or name]
[p]permissions addserverrule allow "playlist start" [voice channel ID or name]
```

Allowing extra roles to use `[p]cleanup`:

```
[p]permissions addserverrule allow cleanup [role ID]
```

Preventing `[p]cleanup` from being used in channels where message history is important:

```
[p]permissions addserverrule deny cleanup [channel ID or mention]
```

CHAPTER

EIGHT

MIGRATING COGS TO V3

First, be sure to read discord.py's migration guide as that covers all of the changes to discord.py that will affect the migration process

8.1 Red as a package

V3 makes Red a package that is installed with `pip`. Please keep this in mind when writing cogs as this affects how imports should be done (for example, to import `pagify` in V2, one would do `from .utils.chat_formatting import pagify`; in V3, this becomes `from redbot.core.utils.chat_formatting import pagify`)

8.2 Cogs as packages

V3 makes cogs into packages. See *Creating cogs for V3* for more on how to create packages for V3.

8.3 Config

Config is V3's replacement for `dataIO`. Instead of fiddling with creating config directories and config files as was done in V2, V3's Config handles that whilst allowing for easy storage of settings on a per-server/member/user/role/channel or global basis. Be sure to check out *Config* for the API docs for Config as well as a tutorial on using Config.

8.4 Bank

Bank in V3 has been split out from Economy. V3 introduces the ability to have a global bank as well as the ability to change the bank name and the name of the currency. Be sure to checkout *Bank* for more on Bank

8.5 Mod Log

V3 introduces Mod Log as an API, thus allowing for cogs to add custom case types that will appear in a server's mod log channel. Be sure to checkout *Mod log* for more on Mod Log'

CHAPTER

NINE

CREATING COGS FOR V3

This guide serves as a tutorial on creating cogs for Red V3. It will cover the basics of setting up a package for your cog and the basics of setting up the file structure. We will also point you towards some further resources that may assist you in the process.

9.1 Getting started

To start off, be sure that you have installed Python 3.7. Open a terminal or command prompt and type `pip install -U git+https://github.com/Cog-Creators/Red-DiscordBot@V3/develop#egg=redbot[test]` (note that if you get an error with this, try again but put `python -m` in front of the command This will install the latest version of V3.

9.2 Setting up a package

To set up a package, we would just need to create a new folder. This should be named whatever you want the cog to be named (for the purposes of this example, we'll call this `mycog`). In this folder, create three files: `__init__.py`, `mycog.py`, and `info.json`. Open the folder in a text editor or IDE (examples include Sublime Text 3, Visual Studio Code, Atom, and PyCharm).

9.3 Creating a cog

With your package opened in a text editor or IDE, open `mycog.py`. In that file, place the following code:

```
from redbot.core import commands

class Mycog(commands.Cog):
    """My custom cog"""

    @commands.command()
    async def mycom(self, ctx):
        """This does stuff!"""
        # Your code will go here
        await ctx.send("I can do stuff!")
```

Open `__init__.py`. In that file, place the following:

```
from .mycog import Mycog

def setup(bot):
    bot.add_cog(Mycog())
```

Make sure that both files are saved.

9.4 Testing your cog

To test your cog, you will need a running instance of V3. Assuming you installed V3 as outlined above, run `redbot-setup` and provide the requested information. Once that's done, run Red by doing `redbot <instance name> --dev` to start Red. Complete the inital setup by providing a valid token and setting a prefix. Once the bot has started up, use the link provided in the console to add it to a server (note that you must have the `Manage Server` (or `Administrator`) permission to add bots to a server). Once it's been added to a server, find the full path to the directory where your cog package is located. In Discord, do `[p]addpath <path_to_folder_containing_package>`, then do `[p]load mycog`. Once the cog is loaded, do `[p]mycom` The bot should respond with `I can do stuff!`. If it did, you have successfully created a cog!

9.5 Additional resources

Be sure to check out the *Migrating Cogs to V3* for some resources on developing cogs for V3. This will also cover differences between V2 and V3 for those who developed cogs for V2.

CHAPTER

TEN

SHARED API KEYS

Red has a central API key storage utilising the core bots config. This allows cog creators to add a single location to store API keys for their cogs which may be shared between other cogs.

There needs to be some consistency between cog creators when using shared API keys between cogs. To help make this easier service should be all **lowercase** and the key names should match the naming convetion of the API being accessed.

Example:

Twitch has a client ID and client secret so a user should be asked to input

```
[p]set api twitch client_id,1234ksdjf client_secret,1234aldlfkd
```

and when accessed in the code it should be done by

```
await self.bot.db.api_tokens.get_raw("twitch", default={"client_id": None, "client_
↪secret": None})
```

Each service has its own dict of key, value pairs for each required key type. If there's only one key required then a name for the key is still required for storing and accessing.

Example:

```
[p]set api youtube api_key,1234ksdjf
```

and when accessed in the code it should be done by

```
await self.bot.db.api_tokens.get_raw("youtube", default={"api_key": None})
```

10.1 Basic Usage

```
class MyCog:
    @commands.command()
    async def youtube(self, ctx, user: str):
        apikey = await self.bot.db.api_tokens.get_raw("youtube", default={"api_key":␣
↪None})
        if apikey["api_key"] is None:
            return await ctx.send("The YouTube API key has not been set.")
        # Use the API key to access content as you normally would
```

CHAPTER

ELEVEN

BANK

Bank has now been separated from Economy for V3. New to bank is support for having a global bank.

11.1 Basic Usage

```
from redbot.core import bank, commands
import discord

class MyCog(commands.Cog):
    @commands.command()
    async def balance(self, ctx, user: discord.Member = None):
        if user is None:
            user = ctx.author
        bal = await bank.get_balance(user)
        currency = await bank.get_currency_name(ctx.guild)
        await ctx.send(
            "{}'s balance is {} {}".format(
                user.display_name, bal, currency
            )
        )
```

11.2 API Reference

11.2.1 Bank

@redbot.core.bank.**cost**(*amount*)

Decorates a coroutine-function or command to have a cost.

If the command raises an exception, the cost will be refunded.

You can intentionally refund by raising *AbortPurchase* (this error will be consumed and not show to users)

Other exceptions will propogate and will be handled by Red's (and/or any other configured) error handling.

class redbot.core.bank.**Account**(*name*, *balance*, *created_at*)

Bases: object

A single account.

This class should ONLY be instantiated by the bank itself.

await redbot.core.bank.**get_balance**(*member*) → int
Get the current balance of a member.

Parameters **member** (*discord.Member*) – The member whose balance to check.

Returns The member's balance

Return type int

await redbot.core.bank.**set_balance**(*member*, *amount*) → int
Set an account balance.

Parameters

- **member** (*discord.Member*) – The member whose balance to set.
- **amount** (*int*) – The amount to set the balance to.

Returns New account balance.

Return type int

Raises

- **ValueError** – If attempting to set the balance to a negative number.
- **BalanceTooHigh** – If attempting to set the balance to a value greater than bank.MAX_BALANCE

await redbot.core.bank.**withdraw_credits**(*member*, *amount*) → int
Remove a certain amount of credits from an account.

Parameters

- **member** (*discord.Member*) – The member to withdraw credits from.
- **amount** (*int*) – The amount to withdraw.

Returns New account balance.

Return type int

Raises

- **ValueError** – If the withdrawal amount is invalid or if the account has insufficient funds.
- **TypeError** – If the withdrawal amount is not an int.

await redbot.core.bank.**deposit_credits**(*member*, *amount*) → int
Add a given amount of credits to an account.

Parameters

- **member** (*discord.Member*) – The member to deposit credits to.
- **amount** (*int*) – The amount to deposit.

Returns The new balance.

Return type int

Raises

- **ValueError** – If the deposit amount is invalid.
- **TypeError** – If the deposit amount is not an int.

await redbot.core.bank.**can_spend**(*member*, *amount*) → bool
Determine if a member can spend the given amount.

Parameters

- **member** (*discord.Member*) – The member wanting to spend.
- **amount** (*int*) – The amount the member wants to spend.

Returns `True` if the member has a sufficient balance to spend the amount, else `False`.

Return type bool

await redbot.core.bank.**transfer_credits**(*from_*, *to*, *amount*)
Transfer a given amount of credits from one account to another.

Parameters

- **from_** (*discord.Member*) – The member to transfer from.
- **to** (*discord.Member*) – The member to transfer to.
- **amount** (*int*) – The amount to transfer.

Returns The new balance of the member gaining credits.

Return type int

Raises

- **ValueError** – If the amount is invalid or if `from_` has insufficient funds.
- **TypeError** – If the amount is not an `int`.

await redbot.core.bank.**wipe_bank**(*guild = None*) → None
Delete all accounts from the bank.

Parameters **guild** (*discord.Guild*) – The guild to clear accounts for. If unsupplied and the bank is per-server, all accounts in every guild will be wiped.

await redbot.core.bank.**get_account**(*member*) → redbot.core.bank.Account
Get the appropriate account for the given user or member.

A member is required if the bank is currently guild specific.

Parameters **member** (`discord.User` or `discord.Member`) – The user whose account to get.

Returns The user's account.

Return type *Account*

await redbot.core.bank.**is_global**() → bool
Determine if the bank is currently global.

Returns `True` if the bank is global, otherwise `False`.

Return type bool

await redbot.core.bank.**set_global**(*global_*) → bool
Set global status of the bank.

Important: All accounts are reset when you switch!

Parameters **global_** (*bool*) – `True` will set bank to global mode.

Returns New bank mode, `True` is global.

Return type bool

Raises **RuntimeError** – If bank is becoming global and a discord.Member was not provided.

await redbot.core.bank.**get_bank_name**(*guild* = *None*) → str
Get the current bank name.

Parameters **guild** (discord.Guild, optional) – The guild to get the bank name for (required if bank is guild-specific).

Returns The bank's name.

Return type str

Raises **RuntimeError** – If the bank is guild-specific and guild was not provided.

await redbot.core.bank.**set_bank_name**(*name*, *guild* = *None*) → str
Set the bank name.

Parameters

- **name** (*str*) – The new name for the bank.
- **guild** (discord.Guild, optional) – The guild to set the bank name for (required if bank is guild-specific).

Returns The new name for the bank.

Return type str

Raises **RuntimeError** – If the bank is guild-specific and guild was not provided.

await redbot.core.bank.**get_currency_name**(*guild* = *None*) → str
Get the currency name of the bank.

Parameters **guild** (discord.Guild, optional) – The guild to get the currency name for (required if bank is guild-specific).

Returns The currency name.

Return type str

Raises **RuntimeError** – If the bank is guild-specific and guild was not provided.

await redbot.core.bank.**set_currency_name**(*name*, *guild* = *None*) → str
Set the currency name for the bank.

Parameters

- **name** (*str*) – The new name for the currency.
- **guild** (discord.Guild, optional) – The guild to set the currency name for (required if bank is guild-specific).

Returns The new name for the currency.

Return type str

Raises **RuntimeError** – If the bank is guild-specific and guild was not provided.

await redbot.core.bank.**get_default_balance**(*guild* = *None*) → int
Get the current default balance amount.

Parameters **guild** (discord.Guild, optional) – The guild to get the default balance for (required if bank is guild-specific).

Returns The bank's default balance.

Return type int

Raises **RuntimeError** – If the bank is guild-specific and guild was not provided.

await redbot.core.bank.**set_default_balance**(*amount*, *guild* = *None*) → int

Set the default balance amount.

Parameters

- **amount** (*int*) – The new default balance.
- **guild** (discord.Guild, optional) – The guild to set the default balance for (required if bank is guild-specific).

Returns The new default balance.

Return type int

Raises

- **RuntimeError** – If the bank is guild-specific and guild was not provided.
- **ValueError** – If the amount is invalid.

exception redbot.core.bank.**AbortPurchase**

Bases: Exception

CHAPTER

TWELVE

BOT

12.1 RedBase

class redbot.core.bot.**RedBase**(**args*, *cli_flags=None*, *bot_dir = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/red-discordbot/checkouts/stable/docs')*, ***kwargs*)

Bases: redbot.core.commands.commands.GroupMixin, discord.ext.commands.bot.BotBase, redbot.core.rpc.RPCMixin

Mixin for the main bot class.

This exists because *Red* inherits from discord.AutoShardedClient, which is something other bot classes may not want to have as a parent class.

register_rpc_handler(*method*)

Registers a method to act as an RPC handler if the internal RPC server is active.

When calling this method through the RPC server, use the naming scheme "cogname__methodname".

Important: All parameters to RPC handler methods must be JSON serializable objects. The return value of handler methods must also be JSON serializable.

Parameters **method** (*coroutine*) – The method to register with the internal RPC server.

unregister_rpc_handler(*method*)

Unregisters an RPC method handler.

This will be called automatically for you on cog unload and will pass silently if the method is not previously registered.

Parameters **method** (*coroutine*) – The method to unregister from the internal RPC server.

add_cog(*cog*)

Adds a "cog" to the bot.

A cog is a class that has its own event listeners and commands.

Parameters **cog** (Cog) – The cog to register to the bot.

Raises

- **TypeError** – The cog does not inherit from Cog.
- **CommandError** – An error happened during loading.

add_command(*command*) → None
Adds a *Command* or its subclasses into the internal list of commands.

This is usually not called, instead the `command()` or `group()` shortcut decorators are used instead.

Parameters **command** (`Command`) – The command to add.

Raises

- **ClientException** – If the command is already registered.
- **TypeError** – If the command passed is not a subclass of *Command*.

add_permissions_hook(*hook*) → None
Add a permissions hook.

Permissions hooks are check predicates which are called before calling *Requires.verify*, and they can optionally return an override: `True` to allow, `False` to deny, and `None` to default to normal behaviour.

Parameters **hook** – A command check predicate which returns `True`, `False` or `None`.

clear_permission_rules(*guild_id*) → None
Clear all permission overrides in a scope.

Parameters **guild_id** (*Optional[int]*) – The guild ID to wipe permission overrides for. If `None`, this will clear all global rules and leave all guild rules untouched.

await embed_requested(*channel*, *user*, *command=None*) → bool
Determine if an embed is requested for a response.

Parameters

- **channel** (`discord.abc.GuildChannel` or `discord.abc.PrivateChannel`) – The channel to check embed settings for.
- **user** (`discord.abc.User`) – The user to check embed settings for.
- **command** – (Optional) the command ran.

Returns `True` if an embed is requested

Return type bool

await get_owner_notification_destinations() → List[discord.abc.Messageable]
Gets the users and channels to send to

await is_admin(*member*)
Checks if a member is an admin of their guild.

await is_automod_immune(*to_check*) → bool
Checks if the user, message, context, or role should be considered immune from automated moderation actions.

This will return `False` in direct messages.

Parameters **to_check** (`discord.Message` or *commands.Context* or `discord.abc.User` or `discord.Role`) – Something to check if it would be immune

Returns `True` if immune

Return type bool

await is_mod(*member*)
Checks if a member is a mod or admin of their guild.

await is_owner(*user*)
Checks if a `User` or `Member` is the owner of this bot.

If an `owner_id` is not set, it is fetched automatically through the use of `application_info()`.

Parameters **user** (`abc.User`) – The user to check for.

Returns Whether the user is the owner.

Return type `bool`

staticmethod list_packages()
Lists packages present in the cogs the folder

await load_extension(*spec*)
Loads an extension.

An extension is a python module that contains commands, cogs, or listeners.

An extension must have a global function, `setup` defined as the entry point on what to do when the extension is loaded. This entry point must have a single argument, the `bot`.

Parameters **name** (`str`) – The extension name to load. It must be dot separated like regular Python imports if accessing a sub-module. e.g. `foo.test` if you want to import `foo/test.py`.

Raises

- **ExtensionNotFound** – The extension could not be imported.
- **ExtensionAlreadyLoaded** – The extension is already loaded.
- **NoEntryPointError** – The extension does not have a setup function.
- **ExtensionFailed** – The extension setup function had an execution error.

await maybe_update_config()
This should be run prior to loading cogs or connecting to discord.

await process_commands(*message*)
Same as base method, but dispatches an additional event for cogs which want to handle normal messages differently to command messages, without the overhead of additional get_context calls per cog.

remove_cog(*cogname*)
Removes a cog from the bot.

All registered commands and event listeners that the cog has registered will be removed as well.

If no cog is found then this method has no effect.

Parameters **name** (`str`) – The name of the cog to remove.

remove_command(*name*) → None
Remove a *Command* or subclasses from the internal list of commands.

This could also be used as a way to remove aliases.

Parameters **name** (`str`) – The name of the command to remove.

Returns The command that was removed. If the name is not valid then `None` is returned instead.

Return type *Command* or subclass

remove_permissions_hook(*hook*) → None
Remove a permissions hook.

Parameters are the same as those in *add_permissions_hook*.

Raises **ValueError** – If the permissions hook has not been added.

staticmethod await send_filtered(*destination*, *filter_mass_mentions=True*, *filter_invite_links=True*, *filter_all_links=False*, ***kwargs*)

This is a convienience wrapper around

discord.abc.Messageable.send

It takes the destination you'd like to send to, which filters to apply (defaults on mass mentions, and invite links) and any other parameters normally accepted by destination.send

This should realistically only be used for responding using user provided input. (unfortunately, including usernames) Manually crafted messages which dont take any user input have no need of this

await send_help_for(*ctx*, *help_for*)

Invokes Red's helpformatter for a given context and object.

await send_to_owners(*content=None*, ***kwargs*)

This sends something to all owners and their configured extra destinations.

This takes the same arguments as discord.abc.Messageable.send

This logs failing sends

await verify_permissions_hooks(*ctx*) → Optional[bool]

Run permissions hooks.

Parameters **ctx** (commands.Context) – The context for the command being invoked.

Returns `False` if any hooks returned `False`, `True` if any hooks return `True` and none returned `False`, `None` otherwise.

Return type Optional[bool]

12.2 Red

class redbot.core.bot.**Red**(**args*, *cli_flags=None*, *bot_dir = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/discordbot/checkouts/stable/docs')*, ***kwargs*)

Bases: *redbot.core.bot.RedBase*, discord.shard.AutoShardedClient

You're welcome Caleb.

await logout()

Logs out of Discord and closes all connections.

await shutdown(*, *restart = False*)

Gracefully quit Red.

The program will exit with code `0` by default.

Parameters **restart** (*bool*) – If `True`, the program will exit with code `26`. If the launcher sees this, it will attempt to restart the bot.

CHAPTER

THIRTEEN

COMMAND CHECK DECORATORS

The following are all decorators for commands, which add restrictions to where and when they can be run.

redbot.core.checks.**bot_has_permissions**(***perms*)
Complain if the bot is missing permissions.

If the user tries to run the command, but the bot is missing the permissions, it will send a message describing which permissions are missing.

This check cannot be overridden by rules.

redbot.core.checks.**has_permissions**(***perms*)
Restrict the command to users with these permissions.

This check can be overridden by rules.

redbot.core.checks.**is_owner**()
Restrict the command to bot owners.

This check cannot be overridden by rules.

redbot.core.checks.**guildowner**()
Restrict the command to the guild owner.

This check can be overridden by rules.

redbot.core.checks.**guildowner_or_permissions**(***perms*)
Restrict the command to the guild owner or users with these permissions.

This check can be overridden by rules.

redbot.core.checks.**admin**()
Restrict the command to users with the admin role.

This check can be overridden by rules.

redbot.core.checks.**admin_or_permissions**(***perms*)
Restrict the command to users with the admin role or these permissions.

This check can be overridden by rules.

redbot.core.checks.**mod**()
Restrict the command to users with the mod role.

This check can be overridden by rules.

redbot.core.checks.**mod_or_permissions**(***perms*)
Restrict the command to users with the mod role or these permissions.

This check can be overridden by rules.

redbot.core.checks.**bot_in_a_guild**()
Deny the command if the bot is not in a guild.

CHAPTER

FOURTEEN

COG MANAGER

class redbot.core.cog_manager.**CogManager**

Bases: object

Directory manager for Red's cogs.

This module allows you to load cogs from multiple directories and even from outside the bot directory. You may also set a directory for downloader to install new cogs to, the default being the `cogs/` folder in the root bot directory.

await add_path(*path*) → None

Add a cog path to current list.

This will ignore duplicates.

Parameters **path** (pathlib.Path or str) – Path to add.

Raises **ValueError** – If `path` does not resolve to an existing directory.

await available_modules() → List[str]

Finds the names of all available modules to load.

await find_cog(*name*) → Optional[_frozen_importlib.ModuleSpec]

Find a cog in the list of available paths.

Parameters **name** (*str*) – Name of the cog to find.

Returns A module spec to be used for specialized cog loading, if found.

Return type Optional[importlib.machinery.ModuleSpec]

await install_path() → pathlib.Path

Get the install path for 3rd party cogs.

Returns The path to the directory where 3rd party cogs are stored.

Return type pathlib.Path

staticmethod invalidate_caches()

Re-evaluate modules in the py cache.

This is an alias for an importlib internal and should be called any time that a new module has been installed to a cog directory.

await paths() → List[pathlib.Path]

Get all currently valid path directories, in order of priority

Returns A list of paths where cog packages can be found. The install path is highest priority, followed by the user-defined paths, and the core path has the lowest priority.

Return type List[pathlib.Path]

await remove_path(*path*) → None
Remove a path from the current paths list.

Parameters **path** (pathlib.Path or str) – Path to remove.

await set_install_path(*path*) → pathlib.Path
Set the install path for 3rd party cogs.

Note: The bot will not remember your old cog install path which means that **all previously installed cogs** will no longer be found.

Parameters **path** (*pathlib.Path*) – The new directory for cog installs.

Returns Absolute path to the new install directory.

Return type pathlib.Path

Raises **ValueError** – If `path` is not an existing directory.

await set_paths(*paths_*)
Set the current paths list.

Parameters **paths_** (list of pathlib.Path) – List of paths to set.

await user_defined_paths() → List[pathlib.Path]
Get a list of user-defined cog paths.

All paths will be absolute and unique, in order of priority.

Returns A list of user-defined paths.

Return type List[pathlib.Path]

CHAPTER

FIFTEEN

COMMANDS PACKAGE

This package acts almost identically to discord.ext.commands; i.e. all of the attributes from discord.py's are also in ours. Some of these attributes, however, have been slightly modified, while others have been added to extend functionlities used throughout the bot, as outlined below.

redbot.core.commands.**command**(*name=None*, *cls=<class 'redbot.core.commands.commands.Command'>*, ***attrs*)

A decorator which transforms an async function into a *Command*.

Same interface as discord.ext.commands.command.

redbot.core.commands.**group**(*name=None*, ***attrs*)

A decorator which transforms an async function into a *Group*.

Same interface as discord.ext.commands.group.

class redbot.core.commands.**Command**(**args*, ***kwargs*)

Bases: redbot.core.commands.commands.CogCommandMixin, discord.ext.commands.core.Command

Command class for Red.

This should not be created directly, and instead via the decorator.

This class inherits from discord.ext.commands.Command. The attributes listed below are simply additions to the ones listed with that class.

checks

A list of check predicates which cannot be overridden, unlike *Requires.checks*.

Type List[coroutine function]

translator

A translator for this command's help docstring.

Type *Translator*

allow_for(*model_id*, *guild_id*) → None

Actively allow this command for the given model.

Parameters

- **model_id** (*Union[int, str]*) – Must be an int if supplying an ID. str is only valid for "default".
- **guild_id** (*int*) – The guild ID to allow this cog or command in. For global rules, use 0.

await can_run(*ctx*, ***, *check_all_parents = False*, *change_permission_state = False*) → bool

Check if this command can be run in the given context.

This function first checks if the command can be run using discord.py's method `discord.ext.commands.Command.can_run`, then will return the result of *`Requires.verify`*.

Keyword Arguments

- **`check_all_parents`** (*`bool`*) – If `True`, this will check permissions for all of this command's parents and its cog as well as the command itself. Defaults to `False`.
- **`change_permission_state`** (*`bool`*) – Whether or not the permission state should be changed as a result of this call. For most cases this should be `False`. Defaults to `False`.

`await can_see`(*ctx*)

Check if this command is visible in the given context.

In short, this will verify whether the user can run the command, and also whether the command is hidden or not.

Parameters **`ctx`** (*`Context`*) – The invocation context to check with.

Returns `True` if this command is visible in the given context.

Return type bool

`clear_rule_for`(*model_id*, *guild_id*) → Tuple[redbot.core.commands.requires.PermState, redbot.core.commands.requires.PermState]

Clear the rule which is currently set for this model.

Parameters

- **`model_id`** (*`Union[int, str]`*) – Must be an `int` if supplying an ID. `str` is only valid for "default".
- **`guild_id`** (*`int`*) – The guild ID. For global rules, use `0`.

`disable_in`(*guild*) → bool

Disable this command in the given guild.

Parameters **`guild`** (*`discord.Guild`*) – The guild to disable the command in.

Returns `True` if the command wasn't already disabled.

Return type bool

`await do_conversion`(*ctx*, *converter*, *argument*, *param*)

Convert an argument according to its type annotation.

Raises **`ConversionFailure`** – If doing the conversion failed.

Returns The converted argument.

Return type Any

`enable_in`(*guild*) → bool

Enable this command in the given guild.

Parameters **`guild`** (*`discord.Guild`*) – The guild to enable the command in.

Returns `True` if the command wasn't already enabled.

Return type bool

`error`(*coro*)

A decorator that registers a coroutine as a local error handler.

A local error handler is an `on_command_error()` event limited to a single command.

The on_command_error event is still dispatched for commands with a dedicated error handler.

Red's global error handler will ignore commands with a registered error handler.

To have red handle specific errors with the default behavior, call `Red.on_command_error` with `unhandled_by_cog` set to True.

Due to how discord.py wraps exceptions, the exception you are expecting here is likely in `error.original` despite that the normal event handler for bot wide command error handling has no such wrapping.

For example:

```
@a_command.error
async def a_command_error_handler(self, ctx, error):

    if isinstance(error.original, MyErrrorType):
        self.log_exception(error.original)
    else:
        await ctx.bot.on_command_error(ctx, error.original, unhandled_
↪by_cog=True)
```

Parameters **coro** (coroutine function) – The coroutine to register as the local error handler.

Raises **discord.ClientException** – The coroutine is not actually a coroutine.

help
Help string for this command.

If the `help` kwarg was passed into the decorator, it will default to that. If not, it will attempt to translate the docstring of the command's callback function.

parents
Returns all parent commands of this command.

This is sorted by the length of `qualified_name` from highest to lowest. If the command has no parents, this will be an empty list.

Type List[*commands.Group*]

class redbot.core.commands.**Group**(**args*, ***kwargs*)
Bases: redbot.core.commands.commands.GroupMixin, redbot.core.commands.commands.Command, redbot.core.commands.commands.CogGroupMixin, discord.ext.commands.core.Group

Group command class for Red.

This class inherits from *Command*, with `GroupMixin` and `discord.ext.commands.Group` mixed in.

class redbot.core.commands.**Context**(***attrs*)
Bases: discord.ext.commands.context.Context

Command invocation context for Red.

All context passed into commands will be of this type.

This class inherits from `discord.ext.commands.Context`.

clean_prefix
The command prefix, but a mention prefix is displayed nicer.

Type str

await embed_colour()
Helper function to get the colour for an embed.

Returns The colour to be used

Return type discord.Colour

await embed_requested()
Simple helper to call bot.embed_requested with logic around if embed permissions are available

Returns `True` if an embed is requested

Return type bool

await maybe_send_embed(*message*) → discord.message.Message
Simple helper to send a simple message to context without manually checking ctx.embed_requested This should only be used for simple messages.

Parameters **message** (`str`) – The string to send

Returns the message which was sent

Return type discord.Message

Raises

- **discord.Forbidden** – see `discord.abc.Messageable.send`
- **discord.HTTPException** – see `discord.abc.Messageable.send`

me
The bot member or user object.

If the context is DM, this will be a `discord.User` object.

Type discord.abc.User

await react_quietly(*reaction*) → bool
Adds a reaction to to the command message. :returns: `True` if adding the reaction succeeded. :rtype: bool

await send(*content=None*, ***kwargs*)
Sends a message to the destination with the content given.

This acts the same as `discord.ext.commands.Context.send`, with one added keyword argument as detailed below in *Other Parameters*.

Parameters **content** (*str*) – The content of the message to send.

Other Parameters

- **filter** (Callable[`str`] -> `str`) – A function which is used to sanitize the `content` before it is sent. Defaults to *`filter_mass_mentions()`*. This must take a single `str` as an argument, and return the sanitized `str`.
- ****kwargs** – See `discord.ext.commands.Context.send`.

Returns The message that was sent.

Return type discord.Message

await send_help(*command=None*)
Send the command help message.

await send_interactive(*messages*, *box_lang = None*, *timeout = 15*) → List[discord.message.Message]
Send multiple messages interactively.

The user will be prompted for whether or not they would like to view the next message, one at a time. They will also be notified of how many messages are remaining on each prompt.

Parameters

- **messages** (*iterable* of *str*) – The messages to send.
- **box_lang** (*str*) – If specified, each message will be contained within a codeblock of this language.
- **timeout** (*int*) – How long the user has to respond to the prompt before it times out. After timing out, the bot deletes its prompt message.

await tick() → bool
Add a tick reaction to the command message.

Returns True if adding the reaction succeeded.

Return type bool

15.1 commands.requires

This module manages the logic of resolving command permissions and requirements. This includes rules which override those requirements, as well as custom checks which can be overriden, and some special checks like bot permissions checks.

class redbot.core.commands.requires.**PrivilegeLevel**
Bases: enum.IntEnum

Enumeration for special privileges.

ADMIN = 3
User has the admin role.

BOT_OWNER = 5
User is a bot owner.

GUILD_OWNER = 4
User is the guild level.

MOD = 2
User has the mod role.

NONE = 1
No special privilege level.

class redbot.core.commands.requires.**PermState**
Bases: enum.Enum

Enumeration for permission states used by rules.

ACTIVE_ALLOW = 1
This command has been actively allowed, default user checks should be ignored.

ACTIVE_DENY = 5
This command has been actively denied, terminate the command chain.

ALLOWED_BY_HOOK = 6
This command has been actively allowed by a permission hook. check validation doesn't need this, but is useful to developers

CAUTIOUS_ALLOW = 4
This command has been actively denied, but there exists a subcommand in the *ACTIVE_ALLOW* state. This occurs when *PASSIVE_ALLOW* and *ACTIVE_DENY* are combined.

DENIED_BY_HOOK = 7
This command has been actively denied by a permission hook check validation doesn't need this, but is useful to developers

NORMAL = 2
No overrides have been set for this command, make determination from default user checks.

PASSIVE_ALLOW = 3
There exists a subcommand in the *ACTIVE_ALLOW* state, continue down the subcommand tree until we either find it or realise we're on the wrong branch.

class redbot.core.commands.requires.**Requires**(*privilege_level*, *user_perms*, *bot_perms*, *checks*)

Bases: object

This class describes the requirements for executing a specific command.

The permissions described include both bot permissions and user permissions.

checks
A list of checks which can be overridden by rules. Use *Command.checks* if you would like them to never be overridden.

Type List[Callable[[*Context*], Union[bool, Awaitable[bool]]]]

privilege_level
The required privilege level (bot owner, admin, etc.) for users to execute the command. Can be None, in which case the *user_perms* will be used exclusively, otherwise, for levels other than bot owner, the user can still run the command if they have the required *user_perms*.

Type *PrivilegeLevel*

ready_event
Event for when this Requires object has had its rules loaded. If permissions is loaded, this should be set when permissions has finished loading rules into this object. If permissions is not loaded, it should be set as soon as the command or cog is added.

Type asyncio.Event

user_perms
The required permissions for users to execute the command. Can be None, in which case the *privilege_level* will be used exclusively, otherwise, it will pass whether the user has the required *privilege_level* _or_ *user_perms*.

Type Optional[discord.Permissions]

bot_perms
The required bot permissions for a command to be executed. This is not overrideable by other conditions.

Type discord.Permissions

DEFAULT = 'default'
The key for the default rule in a rules dict.

GLOBAL = 0
Should be used in place of a guild ID when setting/getting global rules.

clear_all_rules(*guild_id*) → None
Clear all rules of a particular scope.

This will preserve the default rule, if set.

Parameters **guild_id** (*int*) – The guild ID to clear rules for. If set to *Requires.GLOBAL*, this will clear all global rules and leave all guild rules untouched.

get_rule(*model*, *guild_id*) → redbot.core.commands.requires.PermState
Get the rule for a particular model.

Parameters

- **model** (*Union[int, str, PermissionModel]*) – The model to get the rule for. `str` is only valid for *Requires.DEFAULT*.
- **guild_id** (*int*) – The ID of the guild for the rule's scope. Set to *Requires.GLOBAL* for a global rule.

Returns The state for this rule. See the *PermState* class for an explanation.

Return type *PermState*

reset() → None
Reset this Requires object to its original state.

This will clear all rules, including defaults. It also resets the *Requires.ready_event*.

set_rule(*model_id*, *rule*, *guild_id*) → None
Set the rule for a particular model.

Parameters

- **model_id** (*Union[str, int]*) – The model to add a rule for. `str` is only valid for *Requires.DEFAULT*.
- **rule** (PermState) – Which state this rule should be set as. See the *PermState* class for an explanation.
- **guild_id** (*int*) – The ID of the guild for the rule's scope. Set to *Requires.GLOBAL* for a global rule.

await verify(*ctx*) → bool
Check if the given context passes the requirements.

This will check the bot permissions, overrides, user permissions and privilege level.

Parameters **ctx** (*"Context"*) – The invkokation context to check with.

Returns `True` if the context passes the requirements.

Return type bool

Raises

- **BotMissingPermissions** – If the bot is missing required permissions to run the command.
- **CommandError** – Propogated from any permissions checks.

CHAPTER

SIXTEEN

CONFIG

Config was introduced in V3 as a way to make data storage easier and safer for all developers regardless of skill level. It will take some getting used to as the syntax is entirely different from what Red has used before, but we believe Config will be extremely beneficial to both cog developers and end users in the long run.

16.1 Basic Usage

```
from redbot.core import Config

class MyCog:
    def __init__(self):
        self.config = Config.get_conf(self, identifier=1234567890)

        self.config.register_global(
            foo=True
        )

    @commands.command()
    async def return_some_data(self, ctx):
        await ctx.send(await self.config.foo())
```

16.2 Tutorial

This tutorial will walk you through how to use Config.

First, you need to import Config:

```
from redbot.core import Config
```

Then, in the class's `__init__` function, you need to get a config instance:

```
class MyCog:
    def __init__(self):
        self.config = Config.get_conf(self, identifier=1234567890)
```

The `identifier` in *`Config.get_conf()`* is used to keep your cog's data separate from that of another cog, and thus should be unique to your cog. For example: if we have two cogs named `MyCog` and their identifier is different, each will have its own data without overwriting the other's data. Note that it is also possible to force registration of a data key before allowing you to get and set data for that key by adding `force_registration=True` after identifier (that defaults to `False` though)

After we've gotten that, we need to register default values:

```
class MyCog:
    def __init__(self):
        self.config = Config.get_conf(self, identifier=1234567890)
        default_global = {
            "foobar": True,
            "foo": {
                "bar": True,
                "baz": False
            }
        }
        default_guild = {
            "blah": [],
            "baz": 1234567890
        }
        self.config.register_global(**default_global)
        self.config.register_guild(**default_guild)
```

As seen in the example above, we can set up our defaults in dicts and then use those in the appropriate `register` function. As seen above, there's *Config.register_global()* and *Config.register_guild()*, but there's also *Config.register_member()*, *Config.register_role()*, *Config.register_user()*, and *Config.register_channel()*. Note that `member` stores based on guild id AND the user's id.

Once we have our defaults registered and we have the object, we can now use those values in various ways:

```
@commands.command()
@checks.admin_or_permissions(manage_guild=True)
async def setbaz(self, ctx, new_value):
    await self.config.guild(ctx.guild).baz.set(new_value)
    await ctx.send("Value of baz has been changed!")

@commands.command()
@checks.is_owner()
async def setfoobar(self, ctx, new_value):
    await self.config.foobar.set(new_value)

@commands.command()
async def checkbaz(self, ctx):
    baz_val = await self.config.guild(ctx.guild).baz()
    await ctx.send("The value of baz is {}".format("True" if baz_val else "False"))
```

Notice a few things in the above examples:

1. Global doesn't have anything in between `self.config` and the variable.
2. Both the getters and setters need to be awaited because they're coroutines.
3. If you're getting the value, the syntax is:

   ```
   self.config.<insert scope here, or nothing if global>.variable_name()
   ```

4. If setting, it's:

   ```
   self.config.<insert scope here, or nothing if global>.variable_name.set(new_value)
   ```

It is also possible to use `async with` syntax to get and set config values. When entering the statement, the config value is retreived, and on exit, it is saved. This puts a safeguard on any code within the `async with` block such that if it breaks from the block in any way (whether it be from `return`, `break`, `continue` or an exception), the value will still be saved.

Important: Only mutable config values can be used in the `async with` statement (namely lists or dicts), and they must be modified *in place* for their changes to be saved.

Here is an example of the `async with` syntax:

```
@commands.command()
async def addblah(self, ctx, new_blah):
    guild_group = self.config.guild(ctx.guild)
    async with guild_group.blah() as blah:
        blah.append(new_blah)
    await ctx.send("The new blah value has been added!")
```

Important: Please note that while you have nothing between `config` and the variable name for global data, you also have the following commands to get data specific to each category.

- *`Config.guild()`* for guild data which takes an object of type `discord.Guild`.
- *`Config.member()`* which takes `discord.Member`.
- *`Config.user()`* which takes `discord.User`.
- *`Config.role()`* which takes `discord.Role`.
- *`Config.channel()`* which takes `discord.TextChannel`.

If you need to wipe data from the config, you want to look at `Group.clear()`, or *`Config.clear_all()`* and similar methods, such as *`Config.clear_all_guilds()`*.

Which one you should use depends on what you want to do.

If you're looking to clear data for a single guild/member/channel/role/user, you want to use `Group.clear()` as that will clear the data only for the specified thing.

If using *`Config.clear_all()`*, it will reset all data everywhere.

There are other methods provided to reset data from a particular scope. For example, *`Config.clear_all_guilds()`* resets all guild data. For member data, you can clear on both a per-guild and guild-independent basis, see *`Config.clear_all_members()`* for more info.

16.3 Advanced Usage

Config makes it extremely easy to organize data that can easily fit into one of the standard categories (global, guild, user etc.) but there may come a time when your data does not work with the existing categories. There are now features within Config to enable developers to work with data how they wish.

This usage guide will cover the following features:

- *`Group.get_raw()`*
- *`Group.set_raw()`*
- *`Group.clear_raw()`*

For this example let's suppose that we're creating a cog that allows users to buy and own multiple pets using the built-in Economy credits:

```
from redbot.core import bank
from redbot.core import Config
from discord.ext import commands


class Pets:
    def __init__(self):
        self.conf = Config.get_conf(self, 1234567890)

        # Here we'll assign some default costs for the pets
        self.conf.register_global(
            dog=100,
            cat=100,
            bird=50
        )
        self.conf.register_user(
            pets={}
        )
```

And now that the cog is set up we'll need to create some commands that allow users to purchase these pets:

```
# continued
    @commands.command()
    async def get_pet(self, ctx, pet_type: str, pet_name: str):
        """
        Purchase a pet.

        Pet type must be one of: dog, cat, bird
        """
        # Now we need to determine what the cost of the pet is and
        # if the user has enough credits to purchase it.

        # We will need to use "get_raw"
        try:
            cost = await self.conf.get_raw(pet_type)
        except KeyError:
            # KeyError is thrown whenever the data you try to access does not
            # exist in the registered defaults or in the saved data.
            await ctx.send("Bad pet type, try again.")
            return
```

After we've determined the cost of the pet we need to check if the user has enough credits and then we'll need to assign a new pet to the user. This is very easily done using the V3 bank API and *Group.set_raw()*:

```
# continued
        if await bank.can_spend(ctx.author, cost):
            await self.conf.user(ctx.author).pets.set_raw(
                pet_name, value={'cost': cost, 'hunger': 0}
            )

            # this is equivalent to doing the following

            pets = await self.conf.user(ctx.author).pets()
            pets[pet_name] = {'cost': cost, 'hunger': 0}
            await self.conf.user(ctx.author).pets.set(pets)
```

Since the pets can get hungry we're gonna need a command that let's pet owners check how hungry their pets are:

```
# continued
    @commands.command()
    async def hunger(self, ctx, pet_name: str):
        try:
            hunger = await self.conf.user(ctx.author).pets.get_raw(pet_name, 'hunger')
        except KeyError:
            # Remember, this is thrown if something in the provided identifiers
            # is not found in the saved data or the defaults.
            await ctx.send("You don't own that pet!")
            return

        await ctx.send("Your pet has {}/100 hunger".format(hunger))
```

We're responsible pet owners here, so we've also got to have a way to feed our pets:

```
# continued
    @commands.command()
    async def feed(self, ctx, pet_name: str, food: int):
        # This is a bit more complicated because we need to check if the pet is
        # owned first.
        try:
            pet = await self.conf.user(ctx.author).pets.get_raw(pet_name)
        except KeyError:
            # If the given pet name doesn't exist in our data
            await ctx.send("You don't own that pet!")
            return

        hunger = pet.get("hunger")

        # Determine the new hunger and make sure it doesn't go negative
        new_hunger = max(hunger - food, 0)

        await self.conf.user(ctx.author).pets.set_raw(
            pet_name, 'hunger', value=new_hunger
        )

        # We could accomplish the same thing a slightly different way
        await self.conf.user(ctx.author).pets.get_attr(pet_name).hunger.set(new_
↪hunger)

        await ctx.send("Your pet is now at {}/100 hunger!".format(new_hunger)
```

Of course, if we're less than responsible pet owners, there are consequences:

```
#continued
    @commands.command()
    async def adopt(self, ctx, pet_name: str, *, member: discord.Member):
        try:
            pet = await self.conf.user(member).pets.get_raw(pet_name)
        except KeyError:
            await ctx.send("That person doesn't own that pet!")
            return

        hunger = pet.get("hunger")
        if hunger < 80:
            await ctx.send("That pet is too well taken care of to be adopted.")
            return
```

(continues on next page)

(continued from previous page)

```
        await self.conf.user(member).pets.clear_raw(pet_name)

        # this is equivalent to doing the following

        pets = await self.conf.user(member).pets()
        del pets[pet_name]
        await self.conf.user(member).pets.set(pets)

        await self.conf.user(ctx.author).pets.set_raw(pet_name, value=pet)
        await ctx.send(
            "Your request to adopt this pet has been granted due to "
            "how poorly it was taken care of."
        )
```

16.4 V2 Data Usage

There has been much conversation on how to bring V2 data into V3 and, officially, we recommend that cog developers make use of the public interface in Config (using the categories as described in these docs) rather than simply copying and pasting your V2 data into V3. Using Config as recommended will result in a much better experience for you in the long run and will simplify cog creation and maintenance.

However.

We realize that many of our cog creators have expressed disinterest in writing converters for V2 to V3 style data. As a result we have opened up config to take standard V2 data and allow cog developers to manipulate it in V3 in much the same way they would in V2. The following examples will demonstrate how to accomplish this.

Warning: By following this method to use V2 data in V3 you may be at risk of data corruption if your cog is used on a bot with multiple shards. USE AT YOUR OWN RISK.

```
from redbot.core import Config


class ExampleCog:
    def __init__(self):
        self.conf = Config.get_conf(self, 1234567890)

        self.data = {}

    async def load_data(self):
        self.data = await self.conf.custom("V2", "V2").all()

    async def save_data(self):
        await self.conf.custom("V2", "V2").set(self.data)


async def setup(bot):
    cog = ExampleCog()
    await cog.load_data()
    bot.add_cog(cog)
```

16.5 API Reference

Important: Before we begin with the nitty gritty API Reference, you should know that there are tons of working code examples inside the bot itself! Simply take a peek inside of the `tests/core/test_config.py` file for examples of using Config in all kinds of ways.

Important: When getting, setting or clearing values in Config, all keys are casted to `str` for you. This includes keys within a `dict` when one is being set, as well as keys in nested dictionaries within that `dict`. For example:

```
>>> conf = Config.get_conf(self, identifier=999)
>>> conf.register_global(foo={})
>>> await conf.foo.set_raw(123, value=True)
>>> await conf.foo()
{'123': True}
>>> await conf.foo.set({123: True, 456: {789: False}})
>>> await conf.foo()
{'123': True, '456': {'789': False}}
```

16.5.1 Config

class redbot.core.config.**Config**(*cog_name*, *unique_identifier*, *driver*, *force_registration* = *False*, *defaults* = *None*)

Bases: `object`

Configuration manager for cogs and Red.

You should always use *get_conf* to instantiate a Config object. Use *get_core_conf* for Config used in the core package.

Important: Most config data should be accessed through its respective group method (e.g. *guild()*) however the process for accessing global data is a bit different. There is no `global` method because global data is accessed by normal attribute access:

```
await conf.foo()
```

cog_name
The name of the cog that has requested a *Config* object.

Type `str`

unique_identifier
Unique identifier provided to differentiate cog data when name conflicts occur.

Type `int`

driver
An instance of a driver that implements *redbot.core.drivers.red_base.BaseDriver*.

force_registration
Determines if Config should throw an error if a cog attempts to access an attribute which has not been previously registered.

Note: **You should use this.** By enabling force registration you give Config the ability to alert you instantly if you've made a typo when attempting to access data.

Type bool

await all_channels() → dict
Get all channel data as a dict.

Note: The return value of this method will include registered defaults for values which have not yet been set.

Returns A dictionary in the form {int: dict} mapping CHANNEL_ID -> data.

Return type dict

await all_guilds() → dict
Get all guild data as a dict.

Note: The return value of this method will include registered defaults for values which have not yet been set.

Returns A dictionary in the form {int: dict} mapping GUILD_ID -> data.

Return type dict

await all_members(*guild* = *None*) → dict
Get data for all members.

If guild is specified, only the data for the members of that guild will be returned. As such, the dict will map MEMBER_ID -> data. Otherwise, the dict maps GUILD_ID -> MEMBER_ID -> data.

Note: The return value of this method will include registered defaults for values which have not yet been set.

Parameters **guild** (discord.Guild, optional) – The guild to get the member data from. Can be omitted if data from every member of all guilds is desired.

Returns A dictionary of all specified member data.

Return type dict

await all_roles() → dict
Get all role data as a dict.

Note: The return value of this method will include registered defaults for values which have not yet been set.

Returns A dictionary in the form {int: dict} mapping ROLE_ID -> data.

Return type dict

await all_users() → dict
Get all user data as a dict.

Note: The return value of this method will include registered defaults for values which have not yet been set.

Returns A dictionary in the form {int: dict} mapping USER_ID -> data.

Return type dict

channel(*channel*) → redbot.core.config.Group
Returns a *Group* for the given channel.

This does not discriminate between text and voice channels.

Parameters **channel** (discord.abc.GuildChannel) – A channel object.

Returns The channel's Group object.

Return type *Group*

await clear_all()
Clear all data from this Config instance.

This resets all data to its registered defaults.

Important: This cannot be undone.

---

await clear_all_channels()
Clear all channel data.

This resets all channel data to its registered defaults.

await clear_all_custom(*group_identifier*)
Clear all custom group data.

This resets all custom group data to its registered defaults.

Parameters **group_identifier** (*str*) – The identifier for the custom group. This is casted to str for you.

await clear_all_globals()
Clear all global data.

This resets all global data to its registered defaults.

await clear_all_guilds()
Clear all guild data.

This resets all guild data to its registered defaults.

await clear_all_members(*guild = None*)
Clear all member data.

This resets all specified member data to its registered defaults.

Parameters **guild** (discord.Guild, optional) – The guild to clear member data from. Omit to clear member data from all guilds.

await clear_all_roles()
Clear all role data.

This resets all role data to its registered defaults.

await clear_all_users()
Clear all user data.

This resets all user data to its registered defaults.

custom(*group_identifier*, **identifiers*)
Returns a *Group* for the given custom group.

Parameters

- **group_identifier** (*str*) – Used to identify the custom group.
- **identifiers** (*str*) – The attributes necessary to uniquely identify an entry in the custom group. These are casted to `str` for you.

Returns The custom group's Group object.

Return type *Group*

classmethod get_conf(*cog_instance*, *identifier*, *force_registration=False*, *cog_name=None*)
Get a Config instance for your cog.

> **Warning:** If you are using this classmethod to get a second instance of an existing Config object for a particular cog, you MUST provide the correct identifier. If you do not, you *will* screw up all other Config instances for that cog.

Parameters

- **cog_instance** – This is an instance of your cog after it has been instantiated. If you're calling this method from within your cog's `__init__`, this is just `self`.
- **identifier** (*int*) – A (hard-coded) random integer, used to keep your data distinct from any other cog with the same name.
- **force_registration** (`bool`, optional) – Should config require registration of data keys before allowing you to get/set values? See *force_registration*.
- **cog_name** (*str, optional*) – Config normally uses `cog_instance` to determine tha name of your cog. If you wish you may pass `None` to `cog_instance` and directly specify the name of your cog here.

Returns A new Config object.

Return type *Config*

classmethod get_core_conf(*force_registration = False*)
Get a Config instance for a core module.

All core modules that require a config instance should use this classmethod instead of *get_conf*.

Parameters **force_registration** (`bool`, optional) – See *force_registration*.

guild(*guild*) → redbot.core.config.Group
Returns a *Group* for the given guild.

Parameters **guild** (*discord.Guild*) – A guild object.

Returns The guild's Group object.

Return type *Group*

init_custom(*group_identifier*, *identifier_count*)
Initializes a custom group for usage. This method must be called first!

member(*member*) → redbot.core.config.Group
Returns a *Group* for the given member.

Parameters **member** (*discord.Member*) – A member object.

Returns The member's Group object.

Return type *Group*

register_channel(**kwargs*)
Register default values on a per-channel level.

See *register_global* for more details.

register_custom(*group_identifier*, **kwargs*)
Registers default values for a custom group.

See *register_global* for more details.

register_global(**kwargs*)
Register default values for attributes you wish to store in *Config* at a global level.

Examples

You can register a single value or multiple values:

```
conf.register_global(
    foo=True
)

conf.register_global(
    bar=False,
    baz=None
)
```

You can also now register nested values:

```
_defaults = {
    "foo": {
        "bar": True,
        "baz": False
    }
}

# Will register `foo.bar` == True and `foo.baz` == False
conf.register_global(
    **_defaults
)
```

You can do the same thing without a `_defaults` dict by using double underscore as a variable name separator:

```
# This is equivalent to the previous example
conf.register_global(
    foo__bar=True,
```

(continues on next page)

(continued from previous page)

```
    foo__baz=False
)
```

register_guild(***kwargs*)
Register default values on a per-guild level.

See *register_global* for more details.

register_member(***kwargs*)
Registers default values on a per-member level.

This means that each user's data is guild-dependent.

See *register_global* for more details.

register_role(***kwargs*)
Registers default values on a per-role level.

See *register_global* for more details.

register_user(***kwargs*)
Registers default values on a per-user level.

This means that each user's data is guild-independent.

See *register_global* for more details.

role(*role*) → redbot.core.config.Group
Returns a *Group* for the given role.

Parameters **role** (*discord.Role*) – A role object.

Returns The role's Group object.

Return type *Group*

user(*user*) → redbot.core.config.Group
Returns a *Group* for the given user.

Parameters **user** (*discord.User*) – A user object.

Returns The user's Group object.

Return type *Group*

16.5.2 Group

class redbot.core.config.**Group**(*identifier_data*, *defaults*, *driver*, *force_registration* = *False*)
Bases: *redbot.core.config.Value*

Represents a group of data, composed of more *Group* or *Value* objects.

Inherits from *Value* which means that all of the attributes and methods available in *Value* are also available when working with a *Group* object.

defaults
All registered default values for this Group.

Type dict

force_registration
Same as *Config.force_registration*.

Type bool

driver
A reference to *Config.driver*.

Type *redbot.core.drivers.red_base.BaseDriver*

__getattr__(*item*) → Union[redbot.core.config.Group, redbot.core.config.Value]
Get an attribute of this group.

This special method is called whenever dot notation is used on this object.

Parameters **item** (*str*) – The name of the attribute being accessed.

Returns A child value of this Group. This, of course, can be another *Group*, due to Config's composite pattern.

Return type *Group* or *Value*

Raises **AttributeError** – If the attribute has not been registered and *force_registration* is set to True.

__init__(*identifier_data*, *defaults*, *driver*, *force_registration* = *False*)
Initialize self. See help(type(self)) for accurate signature.

all() → redbot.core.config._ValueCtxManager[typing.Dict[str, typing.Any]][Dict[str, Any]]
Get a dictionary representation of this group's data.

The return value of this method can also be used as an asynchronous context manager, i.e. with async with syntax.

Note: The return value of this method will include registered defaults for values which have not yet been set.

Returns All of this Group's attributes, resolved as raw data values.

Return type dict

await clear_raw(**nested_path*)
Allows a developer to clear data as if it was stored in a standard Python dictionary.

For example:

```
await conf.clear_raw("foo", "bar")

# is equivalent to

data = {"foo": {"bar": None}}
del data["foo"]["bar"]
```

Parameters **nested_path** (*Any*) – Multiple arguments that mirror the arguments passed in for nested dict access. These are casted to str for you.

get_attr(*item*)
Manually get an attribute of this Group.

This is available to use as an alternative to using normal Python attribute access. It may be required if you find a need for dynamic attribute access.

Example

A possible use case:

```
@commands.command()
async def some_command(self, ctx, item: str):
    user = ctx.author

    # Where the value of item is the name of the data field in Config
    await ctx.send(await self.conf.user(user).get_attr(item).foo())
```

Parameters **item** (*str*) – The name of the data field in *Config*. This is casted to `str` for you.

Returns The attribute which was requested.

Return type *Value* or *Group*

await get_raw(**nested_path*, *default=Ellipsis*)

Allows a developer to access data as if it was stored in a standard Python dictionary.

For example:

```
d = await conf.get_raw("foo", "bar")

# is equivalent to

data = {"foo": {"bar": "baz"}}
d = data["foo"]["bar"]
```

Note: If retreiving a sub-group, the return value of this method will include registered defaults for values which have not yet been set.

Parameters

- **nested_path** (*str*) – Multiple arguments that mirror the arguments passed in for nested dict access. These are casted to `str` for you.
- **default** – Default argument for the value attempting to be accessed. If the value does not exist the default will be returned.

Returns The value of the path requested.

Return type Any

Raises **KeyError** – If the value does not exist yet in Config's internal storage.

is_group(*item*) → bool

A helper method for *__getattr__*. Most developers will have no need to use this.

Parameters **item** (*Any*) – See *__getattr__*.

is_value(*item*) → bool

A helper method for *__getattr__*. Most developers will have no need to use this.

Parameters **item** (*Any*) – See *__getattr__*.

nested_update(*current*, *defaults* = *Ellipsis*) → Dict[str, Any]
Robust updater for nested dictionaries

If no defaults are passed, then the instance attribute 'defaults' will be used.

await set(*value*)
Set the value of the data elements pointed to by *identifiers*.

Example

```
# Sets global value "foo" to False
await conf.foo.set(False)

# Sets guild specific value of "bar" to True
await conf.guild(some_guild).bar.set(True)
```

Parameters **value** – The new literal value of this attribute.

await set_raw(**nested_path*, *value*)
Allows a developer to set data as if it was stored in a standard Python dictionary.

For example:

```
await conf.set_raw("foo", "bar", value="baz")

# is equivalent to

data = {"foo": {"bar": None}}
data["foo"]["bar"] = "baz"
```

Parameters

- **nested_path** (*Any*) – Multiple arguments that mirror the arguments passed in for nested `dict` access. These are casted to `str` for you.
- **value** – The value to store.

16.5.3 Value

class redbot.core.config.**Value**(*identifier_data*, *default_value*, *driver*)
Bases: `object`

A singular "value" of data.

identifiers
This attribute provides all the keys necessary to get a specific data element from a json document.

Type Tuple[str]

default
The default value for the data element that *identifiers* points at.

driver
A reference to *Config.driver*.

Type *redbot.core.drivers.red_base.BaseDriver*

__call__(*default=Ellipsis*) → redbot.core.config._ValueCtxManager[typing.Any][Any]
Get the literal value of this data element.

Each *Value* object is created by the *Group.__getattr__* method. The "real" data of the *Value* object is accessed by this method. It is a replacement for a `get()` method.

The return value of this method can also be used as an asynchronous context manager, i.e. with `async with` syntax. This can only be used on values which are mutable (namely lists and dicts), and will set the value with its changes on exit of the context manager.

Example

```
foo = await conf.guild(some_guild).foo()

# Is equivalent to this

group_obj = conf.guild(some_guild)
value_obj = group_obj.foo
foo = await value_obj()
```

Important: This is now, for all intents and purposes, a coroutine.

Parameters **default** (object, optional) – This argument acts as an override for the registered default provided by *default*. This argument is ignored if its value is `None`.

Returns A coroutine object mixed in with an async context manager. When awaited, this returns the raw data value. When used in `async with` syntax, on gets the value on entrance, and sets it on exit.

Return type awaitable mixed with asynchronous context manager

await clear()
Clears the value from record for the data element pointed to by *identifiers*.

await set(*value*)
Set the value of the data elements pointed to by *identifiers*.

Example

```
# Sets global value "foo" to False
await conf.foo.set(False)

# Sets guild specific value of "bar" to True
await conf.guild(some_guild).bar.set(True)
```

Parameters **value** – The new literal value of this attribute.

16.6 Driver Reference

redbot.core.drivers.**get_driver**(*type*, **args*, ***kwargs*)
Selectively import/load driver classes based on the selected type. This is required so that dependencies can differ

between installs (e.g. so that you don't need to install a mongo dependency if you will just be running a json data backend).

Note: See the respective classes for information on what `args` and `kwargs` should be.

Parameters

- **type** (*str*) – One of: json, mongo
- **args** – Dependent on driver type.
- **kwargs** – Dependent on driver type.

Returns Subclass of *red_base.BaseDriver*.

class redbot.core.drivers.**BackendType**

Bases: enum.Enum

An enumeration.

16.6.1 Base Driver

class redbot.core.drivers.red_base.**BaseDriver**(*cog_name*, *identifier*)

Bases: object

await clear(*identifier_data*)

Clears out the value specified by the given identifiers.

Equivalent to using `del` on a dict.

Parameters **identifier_data** –

await get(*identifier_data*)

Finds the value indicate by the given identifiers.

Parameters **identifier_data** –

Returns Stored value.

Return type Any

get_config_details()

Asks users for additional configuration information necessary to use this config driver.

Returns

Return type Dict of configuration details.

await set(*identifier_data*, *value=None*)

Sets the value of the key indicated by the given identifiers.

Parameters

- **identifier_data** –
- **value** – Any JSON serializable python object.

16.6.2 JSON Driver

class redbot.core.drivers.red_json.**JSON**(*cog_name*, *identifier*, ***, *data_path_override = None*, *file_name_override = 'settings.json'*)

Bases: *redbot.core.drivers.red_base.BaseDriver*

Subclass of *red_base.BaseDriver*.

file_name
The name of the file in which to store JSON data.

data_path
The path in which to store the file indicated by *file_name*.

await clear(*identifier_data*)
Clears out the value specified by the given identifiers.

Equivalent to using `del` on a dict.

Parameters **identifier_data** –

await get(*identifier_data*)
Finds the value indicate by the given identifiers.

Parameters **identifier_data** –

Returns Stored value.

Return type Any

get_config_details()
Asks users for additional configuration information necessary to use this config driver.

Returns

Return type Dict of configuration details.

await set(*identifier_data*, *value=None*)
Sets the value of the key indicated by the given identifiers.

Parameters

- **identifier_data** –
- **value** – Any JSON serializable python object.

16.6.3 Mongo Driver

class redbot.core.drivers.red_mongo.**Mongo**(*cog_name*, *identifier*, ***kwargs*)

Bases: *redbot.core.drivers.red_base.BaseDriver*

Subclass of *red_base.BaseDriver*.

await clear(*identifier_data*)
Clears out the value specified by the given identifiers.

Equivalent to using `del` on a dict.

Parameters **identifier_data** –

db
Gets the mongo database for this cog's name.

> **Warning:** Right now this will cause a new connection to be made every time the database is accessed. We will want to create a connection pool down the line to limit the number of connections.

Returns PyMongo Database object.

await get(*identifier_data*)
Finds the value indicate by the given identifiers.

Parameters **identifier_data** –

Returns Stored value.

Return type Any

get_collection(*category*) → pymongo.collection.Collection
Gets a specified collection within the PyMongo database for this cog.

Unless you are doing custom stuff `category` should be one of the class attributes of `core.config.Config`.

Parameters **category** (*str*) –

Returns PyMongo collection object.

await set(*identifier_data*, *value=None*)
Sets the value of the key indicated by the given identifiers.

Parameters

- **identifier_data** –
- **value** – Any JSON serializable python object.

CHAPTER

SEVENTEEN

DATA MANAGER

Data manager is a module that handles all the information necessary to bootstrap the bot into a state where more abstract data management systems can take over.

redbot.core.data_manager.**create_temp_config**()
Creates a default instance for Red, so it can be ran without creating an instance.

> **Warning:** The data of this instance will be removed on next system restart.

redbot.core.data_manager.**load_basic_configuration**(*instance_name_*)
Loads the basic bootstrap configuration necessary for *Config* to know where to store or look for data.

> **Important:** It is necessary to call this function BEFORE getting any *Config* objects!

Parameters **instance_name_** (*str*) – The instance name given by CLI argument and created during redbot setup.

redbot.core.data_manager.**cog_data_path**(*cog_instance=None*, *raw_name = None*) → pathlib.Path
Gets the base cog data path. If you want to get the folder with which to store your own cog's data please pass in an instance of your cog class.

Either `cog_instance` or `raw_name` will be used, not both.

Parameters

- **cog_instance** – The instance of the cog you wish to get a data path for. If calling from a command or method of your cog, this should be `self`.
- **raw_name** (*str*) – The name of the cog to get a data path for.

Returns If `cog_instance` is provided it will return a path to a folder dedicated to a given cog. Otherwise it will return a path to the folder that contains data for all cogs.

Return type pathlib.Path

redbot.core.data_manager.**bundled_data_path**(*cog_instance*) → pathlib.Path
Get the path to the "data" directory bundled with this cog.

The bundled data folder must be located alongside the `.py` file which contains the cog class.

> **Important:** You should *NEVER* write to this directory.

Parameters **cog_instance** – An instance of your cog. If calling from a command or method of your cog, this should be `self`.

Returns Path object to the bundled data folder.

Return type pathlib.Path

Raises **FileNotFoundError** – If no bundled data folder exists.

redbot.core.data_manager.**storage_details**() → dict
Gets any details necessary for config drivers to load.

These are set on setup.

Returns

Return type dict

redbot.core.data_manager.**storage_type**() → str
Gets the storage type as a string.

Returns

Return type str

CHAPTER

EIGHTEEN

DOWNLOADER FRAMEWORK

18.1 Info.json

The optional info.json file may exist inside every package folder in the repo, as well as in the root of the repo. The following sections describe the valid keys within an info file (and maybe how the Downloader cog uses them).

18.1.1 Keys common to both repo and cog info.json (case sensitive)

- `author` (list of strings) - list of names of authors of the cog or repo.
- `description` (string) - A long description of the cog or repo. For cogs, this is displayed when a user executes `!cog info`.
- `install_msg` (string) - The message that gets displayed when a cog is installed or a repo is added

> **Tip:** You can use the `[p]` key in your string to use the prefix used for installing.

- `short` (string) - A short description of the cog or repo. For cogs, this info is displayed when a user executes `!cog list`

18.1.2 Keys specific to the cog info.json (case sensitive)

- `min_bot_version` (string) - Min version number of Red in the format `MAJOR.MINOR.MICRO`
- `max_bot_version` (string) - Max version number of Red in the format `MAJOR.MINOR.MICRO`, if `min_bot_version` is newer than `max_bot_version`, `max_bot_version` will be ignored
- `hidden` (bool) - Determines if a cog is visible in the cog list for a repo.
- `disabled` (bool) - Determines if a cog is available for install.
- `required_cogs` (map of cogname to repo URL) - A map of required cogs that this cog depends on. Downloader will not deal with this functionality but it may be useful for other cogs.
- `requirements` (list of strings) - list of required libraries that are passed to pip on cog install. `SHARED_LIBRARIES` do NOT go in this list.
- `tags` (list of strings) - A list of strings that are related to the functionality of the cog. Used to aid in searching.
- `type` (string) - Optional, defaults to `COG`. Must be either `COG` or `SHARED_LIBRARY`. If `SHARED_LIBRARY` then `hidden` will be `True`.

18.2 API Reference

18.2.1 Installable

class redbot.cogs.downloader.installable.**Installable**(*location*)

Bases: redbot.cogs.downloader.json_mixins.RepoJSONMixin

Base class for anything the Downloader cog can install.

- Modules
- Repo Libraries
- Other stuff?

The attributes of this class will mostly come from the installation's info.json.

repo_name
Name of the repository which this package belongs to.

Type str

author
Name(s) of the author(s).

Type tuple of str, optional

bot_version
The minimum bot version required for this installation. Right now this is always 3.0.0.

Type tuple of int

min_python_version
The minimum python version required for this cog. This field will not apply to repo info.json's.

Type tuple of int

hidden
Whether or not this cog will be hidden from the user when they use *Downloader*'s commands.

Type bool

required_cogs
In the form {cog_name : repo_url}, these are cogs which are required for this installation.

Type dict

requirements
Required libraries for this installation.

Type tuple of str

tags
List of tags to assist in searching.

Type tuple of str

type
The type of this installation, as specified by InstallationType.

Type int

await copy_to(*target_dir*) → bool
Copies this cog/shared_lib to the given directory. This will overwrite any files in the target directory.

Parameters **target_dir** (*pathlib.Path*) – The installation directory to install to.

Returns Status of installation

Return type bool

name
The name of this package.

Type str

18.2.2 Repo

class redbot.cogs.downloader.repo_manager.**Repo**(*name*, *url*, *branch*, *folder_path*, *available_modules = ()*, *loop = None*)
Bases: redbot.cogs.downloader.json_mixins.RepoJSONMixin

available_cogs
All available cogs in this Repo.

This excludes hidden or shared packages.

Type tuple of *installable*

available_libraries
All available shared libraries in this Repo.

Type tuple of *installable*

await clone() → Tuple[str]
Clone a new repo.

Returns All available module names from this repo.

Return type tuple of str

await current_branch() → str
Determine the current branch using git commands.

Returns The current branch name.

Return type str

await current_commit(*branch = None*) → str
Determine the current commit hash of the repo.

Parameters **branch** (str, optional) – Override for repo's branch attribute.

Returns The requested commit hash.

Return type str

await current_url(*folder = None*) → str
Discovers the FETCH URL for a Git repo.

Parameters **folder** (*pathlib.Path*) – The folder to search for a URL.

Returns The FETCH URL.

Return type str

Raises ***NoRemoteURL*** – When the folder does not contain a git repo with a FETCH URL.

await hard_reset(*branch = None*) → None
Perform a hard reset on the current repo.

Parameters **branch** (str, optional) – Override for repo branch attribute.

await install_cog(*cog*, *target_dir*) → bool
Install a cog to the target directory.

Parameters

- **cog** (Installable) – The package to install.
- **target_dir** (*pathlib.Path*) – The target directory for the cog installation.

Returns The success of the installation.

Return type bool

await install_libraries(*target_dir*, *req_target_dir*, *libraries = ()*) → bool
Install shared libraries to the target directory.

If `libraries` is not specified, all shared libraries in the repo will be installed.

Parameters

- **target_dir** (*pathlib.Path*) – Directory to install shared libraries to.
- **req_target_dir** (*pathlib.Path*) – Directory to install shared library requirements to.
- **libraries** (tuple of *Installable*) – A subset of available libraries.

Returns The success of the installation.

Return type bool

await install_raw_requirements(*requirements*, *target_dir*) → bool
Install a list of requirements using pip.

Parameters

- **requirements** (tuple of str) – List of requirement names to install via pip.
- **target_dir** (*pathlib.Path*) – Path to directory where requirements are to be installed.

Returns Success of the installation

Return type bool

await install_requirements(*cog*, *target_dir*) → bool
Install a cog's requirements.

Requirements will be installed via pip directly into `target_dir`.

Parameters

- **cog** (Installable) – Cog for which to install requirements.
- **target_dir** (*pathlib.Path*) – Path to directory where requirements are to be installed.

Returns Success of the installation.

Return type bool

await update() -> (<*class 'str'*>, <*class 'str'*>)
Update the current branch of this repo.

Returns :py:code‘(old commit hash, new commit hash)‘

Return type tuple of str

18.2.3 Repo Manager

class redbot.cogs.downloader.repo_manager.**RepoManager**
Bases: object

await add_repo(*url*, *name*, *branch = None*) → redbot.cogs.downloader.repo_manager.Repo
Add and clone a git repository.

Parameters

- **url** (*str*) – URL to the git repository.
- **name** (*str*) – Internal name of the repository.
- **branch** (*str*) – Name of the default branch to checkout into.

Returns New Repo object representing the cloned repository.

Return type *Repo*

await delete_repo(*name*)
Delete a repository and its folders.

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

Raises ***MissingGitRepo*** – If the repo does not exist.

get_all_repo_names() → Tuple[str]
Get all repo names.

Returns

Return type tuple of str

get_repo(*name*) → Optional[redbot.cogs.downloader.repo_manager.Repo]
Get a Repo object for a repository.

Parameters **name** (*str*) – The name of the repository to retrieve.

Returns Repo object for the repository, if it exists.

Return type *Repo* or None

await update_all_repos() → MutableMapping[redbot.cogs.downloader.repo_manager.Repo, Tuple[str, str]]
Call *Repo.update* on all repositories.

Returns A mapping of *Repo* objects that received new commits to a tuple of str containing old and new commit hashes.

Return type dict

18.2.4 Exceptions

exception redbot.cogs.downloader.errors.**DownloaderException**
Bases: Exception

Base class for Downloader exceptions.

exception redbot.cogs.downloader.errors.**GitException**
Bases: *redbot.cogs.downloader.errors.DownloaderException*

Generic class for git exceptions.

exception redbot.cogs.downloader.errors.**InvalidRepoName**
Bases: *redbot.cogs.downloader.errors.DownloaderException*

Throw when a repo name is invalid. Check the message for a more detailed reason.

exception redbot.cogs.downloader.errors.**ExistingGitRepo**
Bases: *redbot.cogs.downloader.errors.DownloaderException*

Thrown when trying to clone into a folder where a git repo already exists.

exception redbot.cogs.downloader.errors.**MissingGitRepo**
Bases: *redbot.cogs.downloader.errors.DownloaderException*

Thrown when a git repo is expected to exist but does not.

exception redbot.cogs.downloader.errors.**CloningError**
Bases: *redbot.cogs.downloader.errors.GitException*

Thrown when git clone returns a non zero exit code.

exception redbot.cogs.downloader.errors.**CurrentHashError**
Bases: *redbot.cogs.downloader.errors.GitException*

Thrown when git returns a non zero exit code attempting to determine the current commit hash.

exception redbot.cogs.downloader.errors.**HardResetError**
Bases: *redbot.cogs.downloader.errors.GitException*

Thrown when there is an issue trying to execute a hard reset (usually prior to a repo update).

exception redbot.cogs.downloader.errors.**UpdateError**
Bases: *redbot.cogs.downloader.errors.GitException*

Thrown when git pull returns a non zero error code.

exception redbot.cogs.downloader.errors.**GitDiffError**
Bases: *redbot.cogs.downloader.errors.GitException*

Thrown when a git diff fails.

exception redbot.cogs.downloader.errors.**NoRemoteURL**
Bases: *redbot.cogs.downloader.errors.GitException*

Thrown when no remote URL exists for a repo.

exception redbot.cogs.downloader.errors.**PipError**
Bases: *redbot.cogs.downloader.errors.DownloaderException*

Thrown when pip returns a non-zero return code.

CHAPTER

NINETEEN

CUSTOM EVENTS

19.1 RPC Server

Red.**on_shutdown**()
Dispatched when the bot begins it's shutdown procedures.

CHAPTER

TWENTY

INTERNATIONALIZATION FRAMEWORK

20.1 Basic Usage

```
from redbot.core import commands
from redbot.core.i18n import Translator, cog_i18n

_ = Translator("ExampleCog", __file__)

@cog_i18n(_)
class ExampleCog:
    """description"""

    @commands.command()
    async def mycom(self, ctx):
        """command description"""
        await ctx.send(_("This is a test command"))
```

20.2 Tutorial

After making your cog, generate a `messages.pot` file

The process of generating this will depend on the operating system you are using

In a command prompt in your cog's package (where yourcog.py is), create a directory called "locales". Then do one of the following:

Windows: `python <your python install path>\Tools\i18n\pygettext.py -D -n -p locales`

Mac: ?

Linux: `pygettext3 -D -n -p locales`

This will generate a messages.pot file with strings to be translated, including docstrings.

20.3 API Reference

`redbot.core.i18n.`**`cog_i18n`**(*translator*)

Get a class decorator to link the translator to this cog.

class redbot.core.i18n.**Translator**(*name*, *file_location*)

Bases: collections.abc.Callable, typing.Generic

Function to get translated strings at runtime.

__call__(*untranslated*) → str

Translate the given string.

This will look for the string in the translator's `.pot` file, with respect to the current locale.

load_translations()

Loads the current translations.

CHAPTER

TWENTYONE

MOD LOG

Mod log has now been separated from Mod for V3.

21.1 Basic Usage

```
from redbot.core import commands, modlog
import discord

class MyCog(commands.Cog):
    @commands.command()
    @checks.admin_or_permissions(ban_members=True)
    async def ban(self, ctx, user: discord.Member, reason: str = None):
        await ctx.guild.ban(user)
        case = await modlog.create_case(
            ctx.bot, ctx.guild, ctx.message.created_at, action="ban",
            user=user, moderator=ctx.author, reason=reason
        )
        await ctx.send("Done. It was about time.")
```

21.2 Registering Case types

To register case types, use an asynchronous `initialize()` method and call it from your setup function:

```
# mycog/mycog.py
from redbot.core import modlog, commands
import discord

class MyCog(commands.Cog):

    async def initialize(self):
        await self.register_casetypes()

    @staticmethod
    async def register_casetypes():
        # Registering a single casetype
        ban_case = {
            "name": "ban",
            "default_setting": True,
            "image": "\N{HAMMER}",
            "case_str": "Ban",
```

(continues on next page)

(continued from previous page)

```
            # audit_type should be omitted if the action doesn't show
            # up in the audit log.
            "audit_type": "ban",
        }
        try:
            await modlog.register_casetype(**ban_case)
        except RuntimeError:
            pass

        # Registering multiple casetypes
        new_types = [
            {
                "name": "hackban",
                "default_setting": True,
                "image": "\N{BUST IN SILHOUETTE}\N{HAMMER}",
                "case_str": "Hackban",
                "audit_type": "ban",
            },
            {
                "name": "kick",
                "default_setting": True,
                "image": "\N{WOMANS BOOTS}",
                "case_str": "Kick",
                "audit_type": "kick"
            }
        ]
        await modlog.register_casetypes(new_types)
```

```
# mycog/__init__.py
from .mycog import MyCog

async def setup(bot):
    cog = MyCog()
    await cog.initialize()
    bot.add_cog(cog)
```

Important: Image should be the emoji you want to represent your case type with.

21.3 API Reference

21.3.1 Mod log

class redbot.core.modlog.**Case**(*bot*, *guild*, *created_at*, *action_type*, *user*, *moderator*, *case_number*, *reason = None*, *until = None*, *channel = None*, *amended_by = None*, *modified_at = None*, *message = None*)

Bases: object

A single mod log case

await edit(*data*)

Edits a case

Parameters **data** (*dict*) – The attributes to change

classmethod await from_json(*mod_channel*, *bot*, *case_number*, *data*, ***kwargs*)
Get a Case object from the provided information

Parameters

- **mod_channel** (*discord.TextChannel*) – The mod log channel for the guild
- **bot** (Red) – The bot's instance. Needed to get the target user
- **case_number** (*int*) – The case's number.
- **data** (*dict*) – The JSON representation of the case to be gotten
- ****kwargs** – Extra attributes for the Case instance which override values in the data dict. These should be complete objects and not IDs, where possible.

Returns The case object for the requested case

Return type *Case*

Raises

- **discord.NotFound** – The user the case is for no longer exists
- **discord.Forbidden** – Cannot read message history to fetch the original message.
- **discord.HTTPException** – A generic API issue

await message_content(*embed = True*)
Format a case message

Parameters **embed** (*bool*) – Whether or not to get an embed

Returns A rich embed or string representing a case message

Return type discord.Embed or str

to_json() → dict
Transform the object to a dict

Returns The case in the form of a dict

Return type dict

class redbot.core.modlog.**CaseType**(*name*, *default_setting*, *image*, *case_str*, *audit_type = None*, *guild = None*)
Bases: object

A single case type

name
The name of the case

Type str

default_setting
Whether the case type should be on (if True) or off (if False) by default

Type bool

image
The emoji to use for the case type (for example, :boot:)

Type str

case_str
The string representation of the case (example: Ban)

Type str

audit_type
The action type of the action as it would appear in the audit log

Type str, optional

classmethod from_json(*name*, *data*, ***kwargs*)

Parameters

- **name** (*str*) – The casetype's name.
- **data** (*dict*) – The JSON data to create an instance from
- ****kwargs** – Values for other attributes of the instance

Returns

Return type *CaseType*

await is_enabled() → bool
Determines if the case is enabled. If the guild is not set, this will always return False

Returns

True if the guild is set and the casetype is enabled for the guild

False if the guild is not set or if the guild is set and the type is disabled

Return type bool

await set_enabled(*enabled*)
Sets the case as enabled or disabled

Parameters **enabled** (*bool*) – True if the case should be enabled, otherwise False

await to_json()
Transforms the case type into a dict and saves it

await redbot.core.modlog.**get_next_case_number**(*guild*) → int
Gets the next case number

Parameters **guild** (discord.Guild) – The guild to get the next case number for

Returns The next case number

Return type int

await redbot.core.modlog.**get_case**(*case_number*, *guild*, *bot*) → redbot.core.modlog.Case
Gets the case with the associated case number

Parameters

- **case_number** (*int*) – The case number for the case to get
- **guild** (*discord.Guild*) – The guild to get the case from
- **bot** (Red) – The bot's instance

Returns The case associated with the case number

Return type *Case*

Raises **RuntimeError** – If there is no case for the specified number

await redbot.core.modlog.**get_all_cases**(*guild*, *bot*) → List[redbot.core.modlog.Case]
Gets all cases for the specified guild

Parameters

- **guild** (discord.Guild) – The guild to get the cases from
- **bot** (Red) – The bot's instance

Returns A list of all cases for the guild

Return type list

await redbot.core.modlog.**get_cases_for_member**(*guild*, *bot*, ***, *member = None*, *member_id = None*) → List[redbot.core.modlog.Case]

Gets all cases for the specified member or member id in a guild.

Parameters

- **guild** (discord.Guild) – The guild to get the cases from
- **bot** (Red) – The bot's instance
- **member** (discord.Member) – The member to get cases about
- **member_id** (*int*) – The id of the member to get cases about

Returns A list of all matching cases.

Return type list

Raises

- **ValueError** – If at least one of member or member_id is not provided
- **discord.Forbidden** – The bot does not have permission to fetch the modlog message which was sent.
- **discord.HTTPException** – Fetching the user failed.

await redbot.core.modlog.**create_case**(*bot*, *guild*, *created_at*, *action_type*, *user*, *moderator = None*, *reason = None*, *until = None*, *channel = None*) → Optional[redbot.core.modlog.Case]

Creates a new case.

This fires an event on_modlog_case_create

Parameters

- **bot** (Red) – The bot object
- **guild** (*discord.Guild*) – The guild the action was taken in
- **created_at** (*datetime*) – The time the action occurred at
- **action_type** (*str*) – The type of action that was taken
- **user** (*Union[discord.User, discord.Member]*) – The user target by the action
- **moderator** (*Optional[Union[discord.User, discord.Member]]*) – The moderator who took the action
- **reason** (*Optional[str]*) – The reason the action was taken
- **until** (*Optional[datetime]*) – The time the action is in effect until
- **channel** (*Optional[discord.TextChannel]*) – The channel the action was taken in

await redbot.core.modlog.**get_casetype**(*name*, *guild = None*) → Optional[redbot.core.modlog.CaseType]

Gets the case type

Parameters

- **name** (*str*) – The name of the case type to get
- **guild** (*Optional[discord.Guild]*) – If provided, sets the case type's guild attribute to this guild

Returns

Return type Optional[*CaseType*]

await redbot.core.modlog.**get_all_casetypes**(*guild = None*) → List[redbot.core.modlog.CaseType]

Get all currently registered case types

Returns A list of case types

Return type list

await redbot.core.modlog.**register_casetype**(*name*, *default_setting*, *image*, *case_str*, *audit_type = None*) → redbot.core.modlog.CaseType

Registers a case type. If the case type exists and there are differences between the values passed and what is stored already, the case type will be updated with the new values

Parameters

- **name** (*str*) – The name of the case
- **default_setting** (*bool*) – Whether the case type should be on (if `True`) or off (if `False`) by default
- **image** (*str*) – The emoji to use for the case type (for example, :boot:)
- **case_str** (*str*) – The string representation of the case (example: Ban)
- **audit_type** (`str`, optional) – The action type of the action as it would appear in the audit log

Returns The case type that was registered

Return type *CaseType*

Raises

- **RuntimeError** – If the case type is already registered
- **TypeError** – If a parameter is missing
- **ValueError** – If a parameter's value is not valid
- **AttributeError** – If the audit_type is not an attribute of `discord.AuditLogAction`

await redbot.core.modlog.**register_casetypes**(*new_types*) → List[redbot.core.modlog.CaseType]

Registers multiple case types

Parameters **new_types** (*list*) – The new types to register

Returns `True` if all were registered successfully

Return type bool

Raises

- **KeyError** –

- **ValueError** –
- **AttributeError** –

See also:

redbot.core.modlog.register_casetype()

await redbot.core.modlog.**get_modlog_channel**(*guild*) → discord.channel.TextChannel
Get the current modlog channel.

Parameters **guild** (discord.Guild) – The guild to get the modlog channel for.

Returns The channel object representing the modlog channel.

Return type discord.TextChannel

Raises **RuntimeError** – If the modlog channel is not found.

await redbot.core.modlog.**set_modlog_channel**(*guild*, *channel*) → bool
Changes the modlog channel

Parameters

- **guild** (discord.Guild) – The guild to set a mod log channel for
- **channel** (discord.TextChannel or None) – The channel to be set as modlog channel

Returns True if successful

Return type bool

await redbot.core.modlog.**reset_cases**(*guild*) → None
Wipes all modlog cases for the specified guild

Parameters **guild** (discord.Guild) – The guild to reset cases for

CHAPTER

TWENTYTWO

RPC

V3 comes default with an internal RPC server that may be used to remotely control the bot in various ways. Cogs must register functions to be exposed to RPC clients. Each of those functions must only take JSON serializable parameters and must return JSON serializable objects.

To enable the internal RPC server you must start the bot with the `--rpc` flag.

22.1 Examples

```
def setup(bot):
    c = Cog()
    bot.add_cog(c)
    bot.register_rpc_handler(c.rpc_method)
```

22.2 Interacting with the RPC Server

The RPC server opens a websocket bound to port `6133` on `127.0.0.1`. This is not configurable for security reasons as broad access to this server gives anyone complete control over your bot. To access the server you must find a library that implements websocket based JSONRPC in the language of your choice.

There are a few built-in RPC methods to note:

- `GET_METHODS` - Returns a list of available RPC methods.
- `GET_METHOD_INFO` - Will return the docstring for an available RPC method. Useful for finding information about the method's parameters and return values.
- `GET_TOPIC` - Returns a list of available RPC message topics.
- `GET_SUBSCRIPTIONS` - Returns a list of RPC subscriptions.
- `SUBSCRIBE` - Subscribes to an available RPC message topic.
- `UNSUBSCRIBE` - Unsubscribes from an RPC message topic.

All RPC methods accept a list of parameters. The built-in methods above expect their parameters to be in list format.

All cog-based methods expect their parameter list to take one argument, a JSON object, in the following format:

```
params = [
    {
        "args": [],  # A list of positional arguments
        "kwargs": {},  # A dictionary of keyword arguments
```

(continues on next page)

(continued from previous page)

```
    }
]

# As an example, here's a call to "get_method_info"
rpc_call("GET_METHOD_INFO", ["get_methods",])

# And here's a call to "core__load"
rpc_call("CORE__LOAD", {"args": [["general", "economy", "downloader"],], "kwargs": {}}
↪)
```

22.3 API Reference

Please see the *redbot.core.bot.RedBase* class for details on the RPC handler register and unregister methods.

CHAPTER

TWENTYTHREE

UTILITY FUNCTIONS

23.1 General Utility

redbot.core.utils.**deduplicate_iterables**(**iterables*)
Returns a list of all unique items in `iterables`, in the order they were first encountered.

redbot.core.utils.**bounded_gather**(**coros_or_futures*, *loop = None*, *return_exceptions = False*, *limit = 4*, *semaphore = None*) → Awaitable[List[Any]]
A semaphore-bounded wrapper to `asyncio.gather()`.

Parameters

- ***coros_or_futures** – The awaitables to run in a bounded concurrent fashion.
- **loop** (*asyncio.AbstractEventLoop*) – The event loop to use for the semaphore and `asyncio.gather()`.
- **return_exceptions** (*bool*) – If true, gather exceptions in the result list instead of raising.
- **limit** (Optional[`int`]) – The maximum number of concurrent tasks. Used when no `semaphore` is passed.
- **semaphore** (Optional[`asyncio.Semaphore`]) – The semaphore to use for bounding tasks. If `None`, create one using `loop` and `limit`.

Raises **TypeError** – When invalid parameters are passed

redbot.core.utils.**bounded_gather_iter**(**coros_or_futures*, *loop = None*, *limit = 4*, *semaphore = None*) → Iterator[Awaitable[Any]]
An iterator that returns tasks as they are ready, but limits the number of tasks running at a time.

Parameters

- ***coros_or_futures** – The awaitables to run in a bounded concurrent fashion.
- **loop** (*asyncio.AbstractEventLoop*) – The event loop to use for the semaphore and `asyncio.gather()`.
- **limit** (Optional[`int`]) – The maximum number of concurrent tasks. Used when no `semaphore` is passed.
- **semaphore** (Optional[`asyncio.Semaphore`]) – The semaphore to use for bounding tasks. If `None`, create one using `loop` and `limit`.

Raises **TypeError** – When invalid parameters are passed

23.2 Chat Formatting

redbot.core.utils.chat_formatting.**bold**(*text*) → str
Get the given text in bold.

Parameters **text** (*str*) – The text to be marked up.

Returns The marked up text.

Return type str

redbot.core.utils.chat_formatting.**bordered**(**columns*, *ascii_border = False*) → str
Get two blocks of text in a borders.

Note: This will only work with a monospaced font.

Parameters

- ***columns** (sequence of str) – The columns of text, each being a list of lines in that column.
- **ascii_border** (*bool*) – Whether or not the border should be pure ASCII.

Returns The bordered text.

Return type str

redbot.core.utils.chat_formatting.**box**(*text*, *lang = ''*) → str
Get the given text in a code block.

Parameters

- **text** (*str*) – The text to be marked up.
- **lang** (str, optional) – The syntax highlighting language for the codeblock.

Returns The marked up text.

Return type str

redbot.core.utils.chat_formatting.**error**(*text*) → str
Get text prefixed with an error emoji.

Returns The new message.

Return type str

redbot.core.utils.chat_formatting.**escape**(*text*, ***, *mass_mentions = False*, *formatting = False*) → str
Get text with all mass mentions or markdown escaped.

Parameters

- **text** (*str*) – The text to be escaped.
- **mass_mentions** (bool, optional) – Set to True to escape mass mentions in the text.
- **formatting** (bool, optional) – Set to True to escpae any markdown formatting in the text.

Returns The escaped text.

Return type str

redbot.core.utils.chat_formatting.**format_perms_list**(*perms*) → str
Format a list of permission names.

This will return a humanized list of the names of all enabled permissions in the provided `discord.Permissions` object.

Parameters **perms** (*discord.Permissions*) – The permissions object with the requested permissions to list enabled.

Returns The humanized list.

Return type str

redbot.core.utils.chat_formatting.**humanize_list**(*items*) → str
Get comma-separted list, with the last element joined with *and*.

This uses an Oxford comma, because without one, items containing the word *and* would make the output difficult to interpret.

Parameters **items** (*Sequence[str]*) – The items of the list to join together.

Raises **IndexError** – An empty sequence was passed

Examples

```
>>> humanize_list(['One', 'Two', 'Three'])
'One, Two, and Three'
>>> humanize_list(['One'])
'One'
```

redbot.core.utils.chat_formatting.**humanize_timedelta**(*, *timedelta* = *None*, *seconds* = *None*) → str
Get a human timedelta representation

redbot.core.utils.chat_formatting.**info**(*text*) → str
Get text prefixed with an info emoji.

Returns The new message.

Return type str

redbot.core.utils.chat_formatting.**inline**(*text*) → str
Get the given text as inline code.

Parameters **text** (*str*) – The text to be marked up.

Returns The marked up text.

Return type str

redbot.core.utils.chat_formatting.**italics**(*text*) → str
Get the given text in italics.

Parameters **text** (*str*) – The text to be marked up.

Returns The marked up text.

Return type str

for ... in redbot.core.utils.chat_formatting.**pagify**(*text, delims = ['\n'], *, priority = False, escape_mass_mentions = True, shorten_by = 8, page_length = 2000*) → Iterator[str]

Generate multiple pages from the given text.

Note: This does not respect code blocks or inline code.

Parameters

- **text** (*str*) – The content to pagify and send.
- **delims** (sequence of str, optional) – Characters where page breaks will occur. If no delimiters are found in a page, the page will break after page_length characters. By default this only contains the newline.

Other Parameters

- **priority** (bool) – Set to True to choose the page break delimiter based on the order of delims. Otherwise, the page will always break at the last possible delimiter.
- **escape_mass_mentions** (bool) – If True, any mass mentions (here or everyone) will be silenced.
- **shorten_by** (int) – How much to shorten each page by. Defaults to 8.
- **page_length** (int) – The maximum length of each page. Defaults to 2000.

Yields str – Pages of the given text.

redbot.core.utils.chat_formatting.**question**(*text*) → str

Get text prefixed with a question emoji.

Returns The new message.

Return type str

redbot.core.utils.chat_formatting.**strikethrough**(*text*) → str

Get the given text with a strikethrough.

Parameters **text** (*str*) – The text to be marked up.

Returns The marked up text.

Return type str

redbot.core.utils.chat_formatting.**underline**(*text*) → str

Get the given text with an underline.

Parameters **text** (*str*) – The text to be marked up.

Returns The marked up text.

Return type str

redbot.core.utils.chat_formatting.**warning**(*text*) → str

Get text prefixed with a warning emoji.

Returns The new message.

Return type str

23.3 Embed Helpers

redbot.core.utils.embed.**randomize_color**(*embed*) → discord.embeds.Embed
Gives the provided embed a random color. There is an alias for this called randomize_color

Parameters **embed** (*discord.Embed*) – The embed to add a color to

Returns The embed with the color set to a random color

Return type discord.Embed

redbot.core.utils.embed.**randomize_colour**(*embed*) → discord.embeds.Embed
Gives the provided embed a random color. There is an alias for this called randomize_color

Parameters **embed** (*discord.Embed*) – The embed to add a color to

Returns The embed with the color set to a random color

Return type discord.Embed

23.4 Reaction Menus

await redbot.core.utils.menus.**menu**(*ctx*, *pages*, *controls*, *message = None*, *page = 0*, *timeout = 30.0*)
An emoji-based menu

Note: All pages should be of the same type

Note: All functions for handling what a particular emoji does should be coroutines (i.e. `async def`). Additionally, they must take all of the parameters of this function, in addition to a string representing the emoji reacted with. This parameter should be the last one, and none of the parameters in the handling functions are optional

Parameters

- **ctx** (`commands.Context`) – The command context
- **pages** (`list` of `str` or `discord.Embed`) – The pages of the menu.
- **controls** (*dict*) – A mapping of emoji to the function which handles the action for the emoji.
- **message** (*discord.Message*) – The message representing the menu. Usually `None` when first opening the menu
- **page** (*int*) – The current page number of the menu
- **timeout** (*float*) – The time (in seconds) to wait for a reaction

Raises **RuntimeError** – If either of the notes above are violated

redbot.core.utils.menus.**start_adding_reactions**(*message*, *emojis*, *loop = None*) → _asyncio.Task
Start adding reactions to a message.

This is a non-blocking operation - calling this will schedule the reactions being added, but the calling code will continue to execute asynchronously. There is no need to await this function.

This is particularly useful if you wish to start waiting for a reaction whilst the reactions are still being added - in fact, this is exactly what *menu* uses to do that.

This spawns a `asyncio.Task` object and schedules it on `loop`. If `loop` omitted, the loop will be retrieved with `asyncio.get_event_loop`.

Parameters

- **message** (*discord.Message*) – The message to add reactions to.
- **emojis** (*Iterable[Union[str, discord.Emoji]]*) – The emojis to react to the message with.
- **loop** (*Optional[asyncio.AbstractEventLoop]*) – The event loop.

Returns The task for the coroutine adding the reactions.

Return type asyncio.Task

23.5 Event Predicates

class redbot.core.utils.predicates.**MessagePredicate**(*predicate*)

Bases: `collections.abc.Callable`, `typing.Generic`

A simple collection of predicates for message events.

These predicates intend to help simplify checks in message events and reduce boilerplate code.

This class should be created through the provided classmethods. Instances of this class are callable message predicates, i.e. they return `True` if a message matches the criteria.

All predicates are combined with *MessagePredicate.same_context()*.

Examples

Waiting for a response in the same channel and from the same author:

```
await bot.wait_for("message", check=MessagePredicate.same_context(ctx))
```

Waiting for a response to a yes or no question:

```
pred = MessagePredicate.yes_or_no(ctx)
await bot.wait_for("message", check=pred)
if pred.result is True:
    # User responded "yes"
    ...
```

Getting a member object from a user's response:

```
pred = MessagePredicate.valid_member(ctx)
await bot.wait_for("message", check=pred)
member = pred.result
```

result

The object which the message content matched with. This is dependent on the predicate used - see each predicate's documentation for details, not every method will assign this attribute. Defaults to `None`.

Type Any

classmethod cancelled(*ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the message is `[p]cancel`.

Parameters

- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as `user` in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod contained_in(*collection*, *ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response is contained in the specified collection.

The index of the response in the `collection` sequence is assigned to the *result* attribute.

Parameters

- **collection** (*Sequence[str]*) – The collection containing valid responses.
- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as `user` in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod equal_to(*value*, *ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response is equal to the specified value.

Parameters

- **value** (*str*) – The value to compare the response with.
- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as `user` in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod greater(*value*, *ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response is greater than the specified value.

Parameters

- **value** (*Union[int, float]*) – The value to compare the response with.

- **ctx** (*Optional[Context]*) – Same as ctx in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as channel in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as user in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod has_role(*ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response refers to a role which the author has.

Assigns the matching discord.Role object to *result*.

One of user or ctx must be supplied. This predicate cannot be used in DM.

Parameters

- **ctx** (*Optional[Context]*) – Same as ctx in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as channel in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as user in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod length_greater(*length*, *ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response's length is greater than the specified length.

Parameters

- **length** (*int*) – The value to compare the response's length with.
- **ctx** (*Optional[Context]*) – Same as ctx in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as channel in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as user in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod length_less(*length*, *ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response's length is less than the specified length.

Parameters

- **length** (*int*) – The value to compare the response's length with.
- **ctx** (*Optional[Context]*) – Same as ctx in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as channel in *same_context()*.

- **user** (*Optional[discord.TextChannel]*) – Same as `user` in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod less(*value*, *ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response is less than the specified value.

Parameters

- **value** (*Union[int, float]*) – The value to compare the response with.
- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as `user` in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod lower_contained_in(*collection*, *ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Same as *contained_in()*, but the response is set to lowercase before matching.

Parameters

- **collection** (*Sequence[str]*) – The collection containing valid lowercase responses.
- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as `user` in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod lower_equal_to(*value*, *ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response *as lowercase* is equal to the specified value.

Parameters

- **value** (*str*) – The value to compare the response with.
- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as `user` in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod positive(*ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response is a positive number.

Assigns the response to *result* as a `float`.

Parameters

- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as `user` in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod regex(*pattern*, *ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response matches the specified regex pattern.

This predicate will use `re.search` to find a match. The resulting `match object` will be assigned to *result*.

Parameters

- **pattern** (Union[`pattern object`, str]) – The pattern to search for in the response.
- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as `user` in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod same_context(*ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the reaction fits the described context.

Parameters

- **ctx** (*Optional[Context]*) – The current invokation context.
- **channel** (*Optional[discord.TextChannel]*) – The channel we expect a message in. If unspecified, defaults to `ctx.channel`. If `ctx` is unspecified too, the message's channel will be ignored.
- **user** (*Optional[discord.TextChannel]*) – The user we expect a message from. If unspecified, defaults to `ctx.author`. If `ctx` is unspecified too, the message's author will be ignored.

Returns The event predicate.

Return type *MessagePredicate*

classmethod valid_float(*ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response is a float.

Assigns the response to *result* as a float.

Parameters

- **ctx** (*Optional[Context]*) – Same as ctx in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as channel in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as user in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod valid_int(*ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response is an integer.

Assigns the response to *result* as an int.

Parameters

- **ctx** (*Optional[Context]*) – Same as ctx in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as channel in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as user in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod valid_member(*ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response refers to a member in the current guild.

Assigns the matching discord.Member object to *result*.

This predicate cannot be used in DM.

Parameters

- **ctx** (*Optional[Context]*) – Same as ctx in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as channel in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as user in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod valid_role(*ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response refers to a role in the current guild.

Assigns the matching discord.Role object to *result*.

This predicate cannot be used in DM.

Parameters

- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as `user` in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod valid_text_channel(*ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the response refers to a text channel in the current guild.

Assigns the matching `discord.TextChannel` object to *result*.

This predicate cannot be used in DM.

Parameters

- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as `user` in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

classmethod yes_or_no(*ctx = None*, *channel = None*, *user = None*) → redbot.core.utils.predicates.MessagePredicate

Match if the message is "yes"/"y" or "no"/"n".

This will assign `True` for *yes*, or `False` for *no* to the *result* attribute.

Parameters

- **ctx** (*Optional[Context]*) – Same as `ctx` in *same_context()*.
- **channel** (*Optional[discord.TextChannel]*) – Same as `channel` in *same_context()*.
- **user** (*Optional[discord.TextChannel]*) – Same as `user` in *same_context()*.

Returns The event predicate.

Return type *MessagePredicate*

class redbot.core.utils.predicates.**ReactionPredicate**(*predicate*)

Bases: `collections.abc.Callable`, `typing.Generic`

A collection of predicates for reaction events.

All checks are combined with *ReactionPredicate.same_context()*.

Examples

Confirming a yes/no question with a tick/cross reaction:

```
from redbot.core.utils.predicates import ReactionPredicate
from redbot.core.utils.menus import start_adding_reactions

msg = await ctx.send("Yes or no?")
start_adding_reactions(msg, ReactionPredicate.YES_OR_NO_EMOJIS)

pred = ReactionPredicate.yes_or_no(msg, ctx.author)
await ctx.bot.wait_for("reaction_add", check=pred)
if pred.result is True:
    # User responded with tick
    ...
else:
    # User responded with cross
    ...
```

Waiting for the first reaction from any user with one of the first 5 letters of the alphabet:

```
from redbot.core.utils.predicates import ReactionPredicate
from redbot.core.utils.menus import start_adding_reactions

msg = await ctx.send("React to me!")
emojis = ReactionPredicate.ALPHABET_EMOJIS[:5]
start_adding_reactions(msg, emojis)

pred = ReactionPredicate.with_emojis(emojis, msg)
await ctx.bot.wait_for("reaction_add", check=pred)
# pred.result is now the index of the letter in `emojis`
```

result
The object which the message content matched with. This is dependent on the predicate used - see each predicate's documentation for details, not every method will assign this attribute. Defaults to `None`.

Type Any

ALPHABET_EMOJIS = ['', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '', '',
A list of all 26 alphabetical letter emojis.

Type List[str]

NUMBER_EMOJIS = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
A list of all single-digit number emojis, 0 through 9.

Type List[str]

YES_OR_NO_EMOJIS = ('', '')
A tuple containing the tick emoji and cross emoji, in that order.

Type Tuple[str, str]

classmethod same_context(*message = None*, *user = None*) → redbot.core.utils.predicates.ReactionPredicate
Match if a reaction fits the described context.

This will ignore reactions added by the bot user, regardless of whether or not `user` is supplied.

Parameters

- **message** (*Optional[discord.Message]*) – The message which we expect a reaction to. If unspecified, the reaction's message will be ignored.
- **user** (*Optional[discord.abc.User]*) – The user we expect to react. If unspecified, the user who added the reaction will be ignored.

Returns The event predicate.

Return type *ReactionPredicate*

classmethod with_emojis(*emojis*, *message* = *None*, *user* = *None*) → redbot.core.utils.predicates.ReactionPredicate

Match if the reaction is one of the specified emojis.

Parameters

- **emojis** (*Sequence[Union[str, discord.Emoji, discord.PartialEmoji]]*) – The emojis of which one we expect to be reacted.
- **message** (*discord.Message*) – Same as `message` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

Returns The event predicate.

Return type *ReactionPredicate*

classmethod yes_or_no(*message* = *None*, *user* = *None*) → redbot.core.utils.predicates.ReactionPredicate

Match if the reaction is a tick or cross emoji.

The emojis used can are in *ReactionPredicate.YES_OR_NO_EMOJIS*.

This will assign `True` for *yes*, or `False` for *no* to the *result* attribute.

Parameters

- **message** (*discord.Message*) – Same as `message` in *same_context()*.
- **user** (*Optional[discord.abc.User]*) – Same as `user` in *same_context()*.

Returns The event predicate.

Return type *ReactionPredicate*

23.6 Mod Helpers

await redbot.core.utils.mod.**check_permissions**(*ctx*, *perms*) → bool

Check if the author has required permissions.

This will always return `True` if the author is a bot owner, or has the `administrator` permission. If `perms` is empty, this will only check if the user is a bot owner.

Parameters

- **ctx** (`Context`) – The command invokation context to check.
- **perms** (*Dict[str, bool]*) – A dictionary mapping permissions to their required states. Valid permission names are those listed as properties of the `discord.Permissions` class.

Returns `True` if the author has the required permissions.

Return type bool

redbot.core.utils.mod.**get_audit_reason**(*author*, *reason* = *None*)

Construct a reason to appear in the audit log.

Parameters

- **author** (*discord.Member*) – The author behind the audit log action.

- **reason** (*str*) – The reason behind the audit log action.

Returns The formatted audit log reason.

Return type str

await redbot.core.utils.mod.**is_admin_or_superior**(*bot*, *obj*)
Same as *is_mod_or_superior* except for admin permissions.

If a message is passed, its author's permissions are checked. If a role is passed, it simply checks if it is the admin role.

Parameters

- **bot** (redbot.core.bot.Red) – The bot object.
- **obj** (discord.Message or discord.Member or discord.Role) – The object to check permissions for.

Returns True if the object has admin permissions.

Return type bool

Raises **TypeError** – If the wrong type of obj was passed.

await redbot.core.utils.mod.**is_mod_or_superior**(*bot*, *obj*)
Check if an object has mod or superior permissions.

If a message is passed, its author's permissions are checked. If a role is passed, it simply checks if it is one of either the admin or mod roles.

Parameters

- **bot** (redbot.core.bot.Red) – The bot object.
- **obj** (discord.Message or discord.Member or discord.Role) – The object to check permissions for.

Returns True if the object has mod permissions.

Return type bool

Raises **TypeError** – If the wrong type of obj was passed.

await redbot.core.utils.mod.**mass_purge**(*messages*, *channel*)
Bulk delete messages from a channel.

If more than 100 messages are supplied, the bot will delete 100 messages at a time, sleeping between each action.

Note: Messages must not be older than 14 days, and the bot must not be a user account.

Parameters

- **messages** (list of discord.Message) – The messages to bulk delete.
- **channel** (*discord.TextChannel*) – The channel to delete messages from.

Raises

- **discord.Forbidden** – You do not have proper permissions to delete the messages or you're not using a bot account.
- **discord.HTTPException** – Deleting the messages failed.

await redbot.core.utils.mod.**slow_deletion**(*messages*)
Delete a list of messages one at a time.

Any exceptions raised when trying to delete the message will be silenced.

Parameters **messages** (iterable of discord.Message) – The messages to delete.

redbot.core.utils.mod.**strfdelta**(*delta*)
Format a timedelta object to a message with time units.

Parameters **delta** (*datetime.timedelta*) – The duration to parse.

Returns A message representing the timedelta with units.

Return type str

23.7 Tunnel

class redbot.core.utils.tunnel.**Tunnel**(*, *sender*, *origin*, *recipient*)
Bases: object

A tunnel interface for messages

This will return None on init if the destination or source + origin pair is already in use, or the existing tunnel object if one exists for the designated parameters

sender
The person who opened the tunnel

Type discord.Member

origin
The channel in which it was opened

Type discord.TextChannel

recipient
The user on the other end of the tunnel

Type discord.User

await communicate(*, *message*, *topic* = *None*, *skip_message_content* = *False*)
Forwards a message.

Parameters

- **message** (discord.Message) – The message to forward
- **topic** (str) – A string to prepend
- **skip_message_content** (bool) – If this flag is set, only the topic will be sent

Returns a pair of ints matching the ids of the message which was forwarded and the last message the bot sent to do that. useful if waiting for reactions.

Return type int, int

Raises **discord.Forbidden** – This should only happen if the user's DMs are disabled the bot can't upload at the origin channel or can't add reactions there.

staticmethod await files_from_attach(*m*) → List[discord.file.File]
makes a list of file objects from a message returns an empty list if none, or if the sum of file sizes is too large for the bot to send

Parameters **m** (discord.Message) – A message to get attachments from

Returns A list of discord.File objects

Return type list of discord.File

staticmethod await files_from_attatch(*m*) → List[discord.file.File]
makes a list of file objects from a message returns an empty list if none, or if the sum of file sizes is too large for the bot to send

Parameters **m** (discord.Message) – A message to get attachments from

Returns A list of discord.File objects

Return type list of discord.File

staticmethod await message_forwarder(*, *destination*, *content = None*, *embed=None*, *files = None*) → List[discord.message.Message]
This does the actual sending, use this instead of a full tunnel if you are using command initiated reactions instead of persistent event based ones

Parameters

- **destination** (*discord.abc.Messageable*) – Where to send
- **content** (*str*) – The message content
- **embed** (*discord.Embed*) – The embed to send
- **files** (*Optional[List[discord.File]]*) – A list of files to send.

Returns The messages sent as a result.

Return type List[discord.Message]

Raises

- **discord.Forbidden** – see discord.abc.Messageable.send
- **discord.HTTPException** – see discord.abc.Messageable.send

23.8 Common Filters

redbot.core.utils.common_filters.**filter_urls**(*to_filter*) → str
Get a string with URLs sanitized.

This will match any URLs starting with these protocols:

- http://
- https://
- ftp://
- sftp://

Parameters **to_filter** (*str*) – The string to filter.

Returns The sanitized string.

Return type str

redbot.core.utils.common_filters.**filter_invites**(*to_filter*) → str
Get a string with discord invites sanitized.

Will match any discord.gg, discordapp.com/invite, or discord.me invite URL.

Parameters **to_filter** (*str*) – The string to filter.

Returns The sanitized string.

Return type str

redbot.core.utils.common_filters.**filter_mass_mentions**(*to_filter*) → str
Get a string with mass mentions sanitized.

Will match any *here* and/or *everyone* mentions.

Parameters **to_filter** (*str*) – The string to filter.

Returns The sanitized string.

Return type str

redbot.core.utils.common_filters.**filter_various_mentions**(*to_filter*) → str
Get a string with role, user, and channel mentions sanitized.

This is mainly for use on user display names, not message content, and should be applied sparingly.

Parameters **to_filter** (*str*) – The string to filter.

Returns The sanitized string.

Return type str

redbot.core.utils.common_filters.**normalize_smartquotes**(*to_normalize*) → str
Get a string with smart quotes replaced with normal ones

Parameters **to_normalize** (*str*) – The string to normalize.

Returns The normalized string.

Return type str

redbot.core.utils.common_filters.**escape_spoilers**(*content*) → str
Get a string with spoiler syntax escaped.

Parameters **content** (*str*) – The string to escape.

Returns The escaped string.

Return type str

redbot.core.utils.common_filters.**escape_spoilers_and_mass_mentions**(*content*) → str
Get a string with spoiler syntax and mass mentions escaped

Parameters **content** (*str*) – The string to escape.

Returns The escaped string.

Return type str

CHAPTER

TWENTYFOUR

V3.1.0 RELEASE NOTES

24.1 Mongo Driver Migration

Due to the required changes of the Mongo driver for Config, all existing Mongo users will need to complete the below instructions to continue to use Mongo after updating to 3.1. This includes **all** users, regardless of any prior migration attempt to a development version of 3.1.

1. Upgrade to 3.1
2. Convert all existing Mongo instances to JSON using the new converters
3. Start each bot instance while using JSON and load any and all cogs you have in order to successfully preserve data.
4. Turn each instance off and convert back to Mongo. **NOTE:** No data is wiped from your Mongo database when converting to JSON. You may want to use a *new* database name when converting back to Mongo in order to not have duplicate data.

24.2 Setup Utility

New commands were introduced to simplify the conversion/editing/removal process both on our end and the users end. Please use `redbot-setup --help` to learn how to use the new features.

Hint: Converting to JSON: `redbot-setup convert <instance_name> json`

Converting to Mongo: `redbot-setup convert <instance_name> mongo`

CHAPTER

TWENTYFIVE

V3.1.0 CHANGELOG

25.1 Audio

- Add Spotify support (#2328)
- Play local folders via text command (#2457)
- Change pause to a toggle (#2461)
- Remove aliases (#2462)
- Add track length restriction (#2465)
- Seek command can now seek to position (#2470)
- Add option for dc at queue end (#2472)
- Emptydisconnect and status refactor (#2473)
- Queue clean and queue clear addition (#2476)
- Fix for audioset status (#2481)
- Playlist download addition (#2482)
- Add songs when search-queuing (#2513)
- Match v2 behavior for channel change (#2521)
- Bot will no longer complain about permissions when trying to connect to user-limited channel, if it has "Move Members" permission (#2525)
- Fix issue on audiostats command when more than 20 servers to display (#2533)
- Fix for prev command display (#2556)
- Fix for localtrack playing (#2557)
- Fix for playlist queue when not playing (#2586)
- Track search and append fixes (#2591)
- DJ role should ask for a role (#2606)

25.2 Core

- Warn on usage of `yaml.load` (#2326)
- New Event dispatch: `on_message_without_command` (#2338)

- Improve output format of cooldown messages (#2412)
- Delete cooldown messages when expired (#2469)
- Fix local blacklist/whitelist management (#2531)
- `[p]set locale` now only accepts actual locales (#2553)
- `[p]listlocales` now displays `en-US` (#2553)
- `redbot --version` will now give you current version of Red (#2567)
- Redesign help and related formatter (#2628)
- Default locale changed from `en` to `en-US` (#2642)
- New command `[p]datapath` that prints the bot's datapath (#2652)

25.3 Config

- Updated Mongo driver to support large guilds (#2536)
- Introduced `init_custom` method on Config objects (#2545)
- We now record custom group primary key lengths in the core config object (#2550)
- Migrated internal UUIDs to maintain cross platform consistency (#2604)

25.4 DataConverter

- It's dead jim (Removal) (#2554)

25.5 discord.py

- No longer vendoring discord.py (#2587)
- Upgraded discord.py dependency to version 1.0.1 (#2587)

25.6 Downloader

- `[p]cog install` will now tell user that cog has to be loaded (#2523)
- The message when libraries fail to install is now formatted (#2576)
- Fixed bug, that caused Downloader to include submodules on cog list (#2590)
- `[p]cog uninstall` allows to uninstall multiple cogs now (#2592)
- `[p]cog uninstall` will now remove cog from installed cogs even if it can't find the cog in install path anymore (#2595)
- `[p]cog install` will not allow to install cogs which aren't suitable for installed version of Red anymore (#2605)

- Cog Developers now have to use `min_bot_version` in form of version string instead of `bot_version` in info.json and they can also use `max_bot_version` to specify maximum version of Red, more in *Downloader Framework*. (#2605)

25.7 Filter

- Filter performs significantly better on large servers. (#2509)

25.8 Launcher

- Fixed extras in the launcher (#2588)

25.9 Mod

- Admins can now decide how many times message has to be repeated before `deleterepeats` removes it (#2437)
- Fix: make `[p]ban [days]` optional as per the doc (#2602)
- Added the command `voicekick` to kick members from a voice channel with optional mod case. (#2639)

25.10 Permissions

- Removed: `p` alias for `permissions` command (#2467)

25.11 Setup Scripts

- `redbot-setup` now uses the click CLI library (#2579)
- `redbot-setup convert` now used to convert between libraries (#2579)
- Backup support for Mongo is currently broken (#2579)

25.12 Streams

- Add support for custom stream alert messages per guild (#2600)
- Add ability to exclude rerun Twitch streams, and note rerun streams in embed status (#2620)

25.13 Tests

- Test for `trivia` cog uses explicitly utf-8 encoding for checking yaml files (#2565)

25.14 Trivia

- Fix of dead image link for Sao Tome and Principe in `worldflags` trivia (#2540)

25.15 Utility Functions

- New: `chat_formatting.humanize_timedelta` (#2412)
- `Tunnel` - Spelling correction of method name - changed `files_from_attatch` to `files_from_attach` (old name is left for backwards compatibility) (#2496)
- `Tunnel` - fixed behavior of `react_close()`, now when tunnel closes message will be sent to other end (#2507)
- `chat_formatting.humanize_list` - Improved error handling of empty lists (#2597)

CHAPTER

TWENTYSIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Y