Red - Discord Bot Documentation

Release 3.0.0a1

Cog Creators

Dec 06, 2017

Installation Guides:

CHAPTER 1

Installing Red on Windows

1.1 Needed Software

- Python - Red needs at least Python 3.5

Attention: Please note that 3.6 has issues on some versions of Windows. If you try using Red with 3.6 and experience issues, uninstall Python 3.6 and install the latest version of Python 3.5

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

1.2 Installing Red

1. Open a command prompt (open Start, search for "command prompt", then click it)
2. Run the appropriate command, depending on if you want audio or not

- No audio: `python -m pip install -U --process-dependency-links Red-DiscordBot`
- Audio: `python -m pip install -U --process-dependency-links Red-DiscordBot[voice]`

- Development version (without audio): `python -m pip install -U --process-dependency-links git+https://github.com/Cog-Creators/Red-DiscordBot@V3/develop#egg=red-discordbot`
- Development version (with audio): `python -m pip install -U --process-dependency-links git+https://github.com/Cog-Creators/Red-DiscordBot@V3/develop#egg=red-discordbot[voice]`

3. Once that has completed, run `redbot-setup` to set up your instance

- 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)

4. Once done setting up the instance, run `redbot <your instance name>` to run Red. It will walk through the initial setup, asking for your token and a prefix

CHAPTER 2

Installing Red on Mac

2.1 Installing pre-requirements

- **Install Brew**
  - In Finder or Spotlight, search for and open terminal. In the window that will open, paste this: `/usr/bin/ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"` and press enter.
- **After the installation, install the required packages by pasting the commands and pressing enter, one-by-one:**
  - `brew install python3 --with-brewed-openssl`
  - `brew install git`
  - `brew install ffmpeg --with-ffplay`
  - `brew install opus`

2.2 Installing Red

Without audio:

```
pip3 install -U --process-dependency-links red-discordbot
```

With audio:

```
pip3 install -U --process-dependency-links red-discordbot[voice]
```

To install the development version (without audio):

```
pip3 install -U --process-dependency-links git+https://github.com/
Cog-Creators/Red-DiscordBot@V3/develop#egg=red-discordbot
```

To install the development version (with audio):

```
pip3 install -U --process-dependency-links git+https://github.com/
Cog-Creators/Red-DiscordBot@V3/develop#egg=red-discordbot[voice]
```

2.3 Setting up an instance

To set up an instance, run `redbot-setup` and follow the steps there, providing the requested information or accepting the defaults. Keep in mind that the instance name will be the one you use when running the bot, so make it something you can remember

2.4 Running Red

Run `redbot <your instance name>` and go through the initial setup (it will ask for the token and a prefix).

CHAPTER 3

Installing Red on Ubuntu 16.04

Warning: For safety reasons, DO NOT install Red with a root user. Instead, make a new one.

3.1 Installing the pre-requirements

```
sudo apt install python3.5-dev python3-pip build-essential libssl-dev libffi-dev git␣
↪ffmpeg libopus-dev unzip -y
```

3.2 Installing the bot

To install without audio:

```
pip3 install -U --process-dependency-links red-discordbot
```

To install with audio:

```
pip3 install -U --process-dependency-links red-discordbot[voice]
```

To install the development version (without audio):

```
pip3 install -U --process-dependency-links git+https://github.com/
Cog-Creators/Red-DiscordBot@V3/develop#egg=red-discordbot
```

To install the development version (with audio):

```
pip3 install -U --process-dependency-links git+https://github.com/
Cog-Creators/Red-DiscordBot@V3/develop#egg=red-discordbot[voice]
```

3.3 Setting up your instance

Run `redbot-setup` and follow the prompts. It will ask first for where you want to store the data (the default is `~/.local/share/Red-DiscordBot`) and will then ask for confirmation of that selection. Next, it will ask you to choose your storage backend (the default here is JSON). It will then ask for a name for your instance. This can be anything as long as it does not contain spaces; however, keep in mind that this is the name you will use to run your bot, and so it should be something you can remember.

3.4 Running Red

Run `redbot <your instance name>` and run through the initial setup. This will ask for your token and a prefix.

CHAPTER 4

Installing Red on Debian Stretch

Warning: For safety reasons, DO NOT install Red with a root user. Instead, make a new one.

4.1 Installing pre-requirements

```
echo "deb http://httpredir.debian.org/debian stretch-backports main contrib non-free"␣
↪>> /etc/apt/sources.list
apt-get update
apt-get install python3.5-dev python3-pip build-essential libssl-dev libffi-dev git␣
↪ffmpeg libopus-dev unzip -y
```

4.2 Installing the bot

To install without audio:

```
pip3 install -U --process-dependency-links red-discordbot
```

To install with audio:

```
pip3 install -U --process-dependency-links red-discordbot[voice]
```

To install the development version (without audio):

```
pip3 install -U --process-dependency-links git+https://github.com/
Cog-Creators/Red-DiscordBot@V3/develop#egg=red-discordbot
```

To install the development version (with audio):

```
pip3 install -U --process-dependency-links git+https://github.com/
Cog-Creators/Red-DiscordBot@V3/develop#egg=red-discordbot[voice]
```

4.3 Setting up your instance

Run `redbot-setup` and follow the prompts. It will ask first for where you want to store the data (the default is `~/.local/share/Red-DiscordBot`) and will then ask for confirmation of that selection. Next, it will ask you to choose your storage backend (the default here is JSON). It will then ask for a name for your instance. This can be anything as long as it does not contain spaces; however, keep in mind that this is the name you will use to run your bot, and so it should be something you can remember.

4.4 Running Red

Run `redbot <your instance name>` and run through the initial setup. This will ask for your token and a prefix.

CHAPTER 5

Installing Red on CentOS 7

5.1 Installing pre-requirements

```
yum -y groupinstall development
yum -y install https://centos7.iuscommunity.org/ius-release.rpm
yum -y install yum-utils wget which python35u python35u-pip python35u-devel openssl-
↪devel libffi-devel git opus-devel
sh -c "$(wget https://gist.githubusercontent.com/mustafaturan/7053900/raw/
↪27f4c8bad3ee2bb0027a1a52dc8501bf1e53b270/latest-ffmpeg-centos6.sh -O -)"
```

5.2 Installing Red

Without audio:

```
pip3 install -U --process-dependency-links red-discordbot
```

With audio:

```
pip3 install -U --process-dependency-links red-discordbot[voice]
```

To install the development version (without audio):

```
pip3 install -U --process-dependency-links git+https://github.com/
Cog-Creators/Red-DiscordBot@V3/develop#egg=red-discordbot
```

To install the development version (with audio):

```
pip3 install -U --process-dependency-links git+https://github.com/
Cog-Creators/Red-DiscordBot@V3/develop#egg=red-discordbot[voice]
```

5.3 Setting up an instance

Run `redbot-setup` and follow the prompts. It will ask first for where you want to store the data (the default is `~/.local/share/Red-DiscordBot`) and will then ask for confirmation of that selection. Next, it will ask you to choose your storage backend (the default here is JSON). It will then ask for a name for your instance. This can be anything as long as it does not contain spaces; however, keep in mind that this is the name you will use to run your bot, and so it should be something you can remember.

5.4 Running Red

Run `redbot <your instance name>` and run through the initial setup. This will ask for your token and a prefix.

CHAPTER 6

Installing Red on Raspbian Stretch

6.1 Installing pre-requirements

```
sudo apt-get install python3.5-dev python3-pip build-essential libssl-dev libffi-dev␣
↪git libav-tools libopus-dev unzip -y
```

6.2 Installing Red

Without audio:

```
pip3 install -U --process-dependency-links red-discordbot
```

With audio:

```
pip3 install -U --process-dependency-links red-discordbot[voice]
```

To install the development version (without audio):

```
pip3 install -U --process-dependency-links git+https://github.com/
Cog-Creators/Red-DiscordBot@V3/develop#egg=red-discordbot
```

To install the development version (with audio):

```
pip3 install -U --process-dependency-links git+https://github.com/
Cog-Creators/Red-DiscordBot@V3/develop#egg=red-discordbot[voice]
```

6.3 Setting up an instance

Run `redbot-setup` and follow the prompts. It will ask first for where you want to store the data (the default is `~/.local/share/Red-DiscordBot`) and will then ask for confirmation of that selection. Next, it will ask you to choose your storage backend (the default here is JSON). It will then ask for a name for your instance. This can be

anything as long as it does not contain spaces; however, keep in mind that this is the name you will use to run your bot, and so it should be something you can remember.

6.4 Running Red

Run `redbot <your instance name>` and run through the initial setup. This will ask for your token and a prefix.

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

CHAPTER 7

Downloader Cog Reference

class redbot.cogs.downloader.downloader.**Downloader**(*bot: redbot.core.bot.Red*)

coroutine cog_install_path()
Get the current cog install path.

Returns The default cog install path.

Return type pathlib.Path

cog_name_from_instance(*instance: object*) → 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: str*, *cog_installable: object = 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

coroutine installed_cogs() → typing.Tuple[redbot.cogs.downloader.installable.Installable]
Get info on installed cogs.

Returns All installed cogs / shared lib directories.

Return type tuple of *Installable*

coroutine is_installed(*cog_name: str*) -> (*<class 'bool'>*, *typing.Union[redbot.cogs.downloader.installable.Installable, NoneType]*)

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 8

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 9

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 3.5 or higher (if you are on Windows, stick with 3.5). Open a terminal or command prompt and type `pip install --process-dependency-links -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 discord.ext import commands

class Mycog:
    """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 migration guide 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 10

Bank

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

10.1 Basic Usage

```
from redbot.core import bank

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

10.2 API Reference

10.2.1 Bank

class redbot.core.bank.**Account**(*name: str*, *balance: int*, *created_at: datetime.datetime*)
A single account.

This class should ONLY be instantiated by the bank itself.

coroutine redbot.core.bank.**get_balance**(*member: discord.member.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

coroutine redbot.core.bank.**set_balance**(*member: discord.member.Member*, *amount: int*) → 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.

coroutine redbot.core.bank.**withdraw_credits**(*member: discord.member.Member*, *amount: int*) → 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.

coroutine redbot.core.bank.**deposit_credits**(*member: discord.member.Member*, *amount: int*) → 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.

coroutine redbot.core.bank.**can_spend**(*member: discord.member.Member*, *amount: int*) → 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

coroutine redbot.core.bank.**transfer_credits**(*from_: discord.member.Member*, *to: discord.member.Member*, *amount: int*)

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.

Return type int

Raises ValueError – If the amount is invalid or if from_ has insufficient funds.

coroutine redbot.core.bank.**wipe_bank**()

Delete all accounts from the bank.

coroutine redbot.core.bank.**get_guild_accounts**(*guild: discord.guild.Guild*) → typing.List[redbot.core.bank.Account]

Get all account data for the given guild.

Parameters **guild** (*discord.Guild*) – The guild to get accounts for.

Returns A list of all guild accounts.

Return type list of *Account*

Raises RuntimeError – If the bank is currently global.

coroutine redbot.core.bank.**get_global_accounts**() → typing.List[redbot.core.bank.Account]

Get all global account data.

Returns A list of all global accounts.

Return type list of *Account*

Raises RuntimeError – If the bank is currently guild specific.

coroutine redbot.core.bank.**get_account**(*member: typing.Union[discord.member.Member, discord.user.User]*) → 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*

coroutine 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

coroutine redbot.core.bank.**set_global**(*global_: bool*) → 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.

coroutine redbot.core.bank.**get_bank_name**(*guild: discord.guild.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.

coroutine redbot.core.bank.**set_bank_name**(*name: str*, *guild: discord.guild.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.

coroutine redbot.core.bank.**get_currency_name**(*guild: discord.guild.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.

coroutine redbot.core.bank.**set_currency_name**(*name: str*, *guild: discord.guild.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.

coroutine redbot.core.bank.**get_default_balance**(*guild: discord.guild.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.

coroutine redbot.core.bank.**set_default_balance**(*amount: int*, *guild: discord.guild.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.

CHAPTER 11

Cog Manager

class redbot.core.cog_manager.**CogManager**(*paths: typing.Tuple[str] = ()*)

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.

coroutine add_path(*path: typing.Union[pathlib.Path, str]*)

Add a cog path to current list.

This will ignore duplicates. Does have a side effect of removing all invalid paths from the saved path list.

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

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

coroutine available_modules() → typing.List[str]

Finds the names of all available modules to load.

coroutine find_cog(*name: str*) → _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.

Return type importlib.machinery.ModuleSpec

Raises `RuntimeError` – If there is no cog with the given name.

coroutine 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

static 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.

coroutine paths() → typing.Tuple[pathlib.Path, ...]
Get all currently valid path directories.

Returns All valid cog paths.

Return type tuple of pathlib.Path

coroutine remove_path(*path: typing.Union[pathlib.Path, str]*) → typing.Tuple[pathlib.Path, ...]
Remove a path from the current paths list.

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

Returns Tuple of new valid paths.

Return type tuple of pathlib.Path

coroutine set_install_path(*path: pathlib.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.

coroutine set_paths(*paths_: typing.List[pathlib.Path]*)
Set the current paths list.

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

CHAPTER 12

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.**load_basic_configuration**(*instance_name_: str*)

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*) → 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.

Parameters **cog_instance** – The instance of the cog you wish 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.**load_bundled_data**(*cog_instance*, *init_location: str*)

This function copies (and overwrites) data from the data/ folder of the installed cog.

Important: This function MUST be called from the setup() function of your cog.

Examples

```
>>> from redbot.core import data_manager
>>>
>>> def setup(bot):
>>>     cog = MyCog()
>>>     data_manager.load_bundled_data(cog, __file__)
>>>     bot.add_cog(cog)
```

Parameters

- **cog_instance** – An instance of your cog class.
- **init_location** (*str*) – The `__file__` attribute of the file where your `setup()` function exists.

redbot.core.data_manager.**bundled_data_path**(*cog_instance*) → pathlib.Path

The "data" directory that has been copied from installed cogs.

Important: You should *NEVER* write to this directory. Data manager will overwrite files in this directory each time *load_bundled_data* is called. You should instead write to the directory provided by *cog_data_path*.

Parameters **cog_instance** –

Returns Path object to the bundled data folder.

Return type pathlib.Path

Raises `FileNotFoundError` – If no bundled data folder exists or if it hasn't been loaded yet.

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 13

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.

13.1 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 config.foo())
```

13.2 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.

13.3 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.

13.3.1 Config

class redbot.core.config.**Config**(*cog_name: str*, *unique_identifier: str*, *driver_spawn: typing.Callable*, *force_registration: bool = False*, *defaults: dict = None*)

Configuration manager for cogs and Red.

You should always use *get_conf* or 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
`str` – The name of the cog that has requested a *Config* object.

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

spawner
A callable object that returns some driver that implements *redbot.core.drivers.red_base.BaseDriver*.

force_registration
`bool` – 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.

coroutine 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

coroutine 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

coroutine all_members(*guild: discord.guild.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

coroutine 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

coroutine 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: discord.channel.TextChannel*) → 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*

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

This resets all data to its registered defaults.

Important: This cannot be undone.

coroutine clear_all_channels()
Clear all channel data.

This resets all channel data to its registered defaults.

coroutine clear_all_globals()
Clear all global data.

This resets all global data to its registered defaults.

coroutine clear_all_guilds()
Clear all guild data.

This resets all guild data to its registered defaults.

coroutine clear_all_members(*guild: discord.guild.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.

coroutine clear_all_roles()
Clear all role data.

This resets all role data to its registered defaults.

coroutine clear_all_users()
Clear all user data.

This resets all user data to its registered defaults.

classmethod get_conf(*cog_instance*, *identifier: int*, *force_registration=False*)
Get a Config instance for your 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*.

Returns A new Config object.

Return type *Config*

classmethod get_core_conf(*force_registration: bool = 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: discord.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*

member(*member: discord.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_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,
```

```
    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: discord.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: discord.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*

13.3.2 Group

class redbot.core.config.**Group**(*identifiers: typing.Tuple[str], defaults: dict, spawner, force_registration: bool = False*)
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
dict – All registered default values for this Group.

force_registration
bool – Same as *Config.force_registration*.

spawner
redbot.core.drivers.red_base.BaseDriver – A reference to *Config.spawner*.

__getattr__(*item: str*) → typing.Union[typing.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.

coroutine all() → dict
Get a dictionary representation of this group's data.

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

coroutine clear()
Wipe all data from this group.

If used on a global group, it will wipe all global data, but not local data.

get_attr(*item: str*, *default=None*, *resolve=True*)
Manually get an attribute of this Group.

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

Note: Use of this method should be avoided wherever possible.

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))
```

Parameters

- **item** (*str*) – The name of the data field in *Config*.
- **default** – This is an optional override to the registered default for this item.
- **resolve** (*bool*) – If this is True this function will return a coroutine that resolves to a "real" data value when awaited. If False, this method acts the same as *__getattr__*.

Returns The attribute which was requested, its type depending on the value of `resolve`.

Return type `types.coroutine` or *Value* or *Group*

is_group (*item: str*) → bool
A helper method for *__getattr__*. Most developers will have no need to use this.

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

is_value (*item: str*) → bool
A helper method for *__getattr__*. Most developers will have no need to use this.

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

coroutine set_attr (*item: str*, *value*)
Set an attribute by its name.

Similar to *get_attr* in the way it can be used to dynamically set attributes by name.

Note: Use of this method should be avoided wherever possible.

Parameters

- **item** (*str*) – The name of the attribute being set.
- **value** – The raw data value to set the attribute as.

13.3.3 Value

class redbot.core.config.**Value** (*identifiers: typing.Tuple[str], default_value, spawner*)
A singular "value" of data.

identifiers
`tuple` of `str` – This attribute provides all the keys necessary to get a specific data element from a json document.

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

spawner
redbot.core.drivers.red_base.BaseDriver – A reference to *Config.spawner*.

__call__ (*default=None*)
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 = conf.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

coroutine 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.

13.4 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*.

13.4.1 Base Driver

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

coroutine get(*identifiers: typing.Tuple[str]*)
Finds the value indicate by the given identifiers.

Parameters **identifiers** – A list of identifiers that correspond to nested dict accesses.

Returns Stored value.

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

Returns Dict of configuration details.

coroutine set(*identifiers: typing.Tuple[str], value*)
Sets the value of the key indicated by the given identifiers.

Parameters

- **identifiers** – A list of identifiers that correspond to nested dict accesses.
- **value** – Any JSON serializable python object.

13.4.2 JSON Driver

class redbot.core.drivers.red_json.**JSON**(*cog_name*, *, *data_path_override: pathlib.Path = None*, *file_name_override: str = 'settings.json'*)
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*.

13.4.3 Mongo Driver

CHAPTER 14

Downloader Framework

14.1 Info.json

The info.json file may exist inside every package folder in the repo, it is optional however. This string describes the valid keys within an info file (and maybe how the Downloader cog uses them).

KEYS (case sensitive):

- `author` (list of strings) - list of names of authors of the cog
- `bot_version` (list of integer) - Min version number of Red in the format `(MAJOR, MINOR, PATCH)`
- `description` (string) - A long description of the cog that appears when a user executes `` `!cog info ``.
- `hidden` (bool) - Determines if a cog is available for install.
- `install_msg` (string) - The message that gets displayed when a cog is installed
- `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.
- `short` (string) - A short description of the cog that appears when a user executes `cog 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`.

14.2 API Reference

14.2.1 Installable

class redbot.cogs.downloader.installable.**Installable**(*location: pathlib.Path*)

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
str – Name of the repository which this package belongs to.

author
tuple of str, optional – Name(s) of the author(s).

bot_version
tuple of int – The minimum bot version required for this installation. Right now this is always 3.0.0.

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

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

requirements
tuple of str – Required libraries for this installation.

tags
tuple of str – List of tags to assist in searching.

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

coroutine copy_to(*target_dir: pathlib.Path*) → 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
str – The name of this package.

14.2.2 Repo

class redbot.cogs.downloader.repo_manager.**Repo**(*name: str, url: str, branch: str, folder_path: pathlib.Path, available_modules: typing.Tuple[redbot.cogs.downloader.installable.Installable] = (), loop: asyncio.events.AbstractEventLoop = None*)

available_cogs
tuple of *installable* – All available cogs in this Repo.

This excludes hidden or shared packages.

available_libraries
tuple of *installable* – All available shared libraries in this Repo.

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

Returns All available module names from this repo.

Return type tuple of str

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

Returns The current branch name.

Return type str

coroutine current_commit(*branch: str = 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

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

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

coroutine install_cog(*cog: redbot.cogs.downloader.installable.Installable*, *target_dir: pathlib.Path*) → 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

coroutine install_libraries(*target_dir: pathlib.Path*, *libraries: typing.Tuple[redbot.cogs.downloader.installable.Installable] = ()*) → 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.
- **libraries** (tuple of *Installable*) – A subset of available libraries.

Returns The success of the installation.

Return type bool

coroutine install_raw_requirements(*requirements: typing.Tuple[str], target_dir: pathlib.Path*) → 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

coroutine install_requirements(*cog: redbot.cogs.downloader.installable.Installable, target_dir: pathlib.Path*) → 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

coroutine 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

14.2.3 Repo Manager

class redbot.cogs.downloader.repo_manager.**RepoManager**(*downloader_config: redbot.core.config.Config*)

coroutine add_repo(*url: str, name: str, branch: str = 'master'*) → 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*

coroutine delete_repo(*name: str*)

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

Returns

Return type tuple of str

get_repo(*name: str*) → typing.Union[redbot.cogs.downloader.repo_manager.Repo, NoneType]
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

coroutine update_all_repos() → typing.MutableMapping[redbot.cogs.downloader.repo_manager.Repo, typing.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

14.2.4 Exceptions

exception redbot.cogs.downloader.errors.**DownloaderException**
Base class for Downloader exceptions.

exception redbot.cogs.downloader.errors.**GitException**
Generic class for git exceptions.

exception redbot.cogs.downloader.errors.**InvalidRepoName**
Throw when a repo name is invalid. Check the message for a more detailed reason.

exception redbot.cogs.downloader.errors.**ExistingGitRepo**
Thrown when trying to clone into a folder where a git repo already exists.

exception redbot.cogs.downloader.errors.**MissingGitRepo**
Thrown when a git repo is expected to exist but does not.

exception redbot.cogs.downloader.errors.**CloningError**
Thrown when git clone returns a non zero exit code.

exception redbot.cogs.downloader.errors.**CurrentHashError**
Thrown when git returns a non zero exit code attempting to determine the current commit hash.

exception redbot.cogs.downloader.errors.**HardResetError**
Thrown when there is an issue trying to execute a hard reset (usually prior to a repo update).

exception redbot.cogs.downloader.errors.**UpdateError**
Thrown when git pull returns a non zero error code.

exception redbot.cogs.downloader.errors.**GitDiffError**
Thrown when a git diff fails.

exception redbot.cogs.downloader.errors.**PipError**
Thrown when pip returns a non-zero return code.

CHAPTER 15

Internationalization Framework

15.1 Basic Usage

```
from discord.ext import commands
from redbot.core.i18n import CogI18n

_ = CogI18n("ExampleCog", __file__)

class ExampleCog:
    """description"""

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

15.2 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 -n -p locales`

Mac: ?

Linux: `pygettext3 -n -p locales`

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

15.3 API Reference

CHAPTER 16

Mod log

Mod log has now been separated from Mod for V3.

16.1 Basic Usage

```
from redbot.core import modlog
import discord

class MyCog:
    @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 = modlog.create_case(
            ctx.guild, ctx.message.created_at, "ban", user,
            ctx.author, reason, until=None, channel=None
        )
        await ctx.send("Done. It was about time.")
```

16.2 Registering Case types

To register a single case type:

```
from redbot.core import modlog
import discord

class MyCog:
    def __init__(self, bot):
        ban_case = {
            "name": "ban",
            "default_setting": True,
```

```
            "image": ":hammer:",
            "case_str": "Ban",
            "audit_type": "ban"
        }
        modlog.register_casetype(**ban_case)
```

To register multiple case types:

```
from redbot.core import modlog
import discord

class MyCog:
    def __init__(self, bot):
        new_types = [
            {
                "name": "ban",
                "default_setting": True,
                "image": ":hammer:",
                "case_str": "Ban",
                "audit_type": "ban"
            },
            {
                "name": "kick",
                "default_setting": True,
                "image": ":boot:",
                "case_str": "Kick",
                "audit_type": "kick"
            }
        ]
        modlog.register_casetypes(new_types)
```

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

16.3 API Reference

16.3.1 Mod log

class redbot.core.modlog.**Case**(*guild: discord.guild.Guild*, *created_at: int*, *action_type: str*, *user: discord.user.User*, *moderator: discord.member.Member*, *case_number: int*, *reason: str = None*, *until: int = None*, *channel: discord.channel.TextChannel = None*, *amended_by: discord.member.Member = None*, *modified_at: int = None*, *message: discord.message.Message = None*)

A single mod log case

coroutine edit(*data: dict*)

Edits a case

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

coroutine from_json(*mod_channel: discord.channel.TextChannel*, *bot: redbot.core.bot.Red*, *data: dict*)

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
- **data** (*dict*) – The JSON representation of the case to be gotten

Returns The case object for the requested case

Return type *Case*

coroutine message_content()
Format a case message

Returns A rich embed representing a case message

Return type discord.Embed

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: str*, *default_setting: bool*, *image: str*, *case_str: str*, *audit_type: str = None*, *guild: discord.guild.Guild = None*)
A single case type

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

classmethod from_json(*data: dict*)

Parameters **data** (*dict*) – The data to create an instance from

Returns

Return type *CaseType*

coroutine 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

coroutine set_enabled(*enabled: bool*)
Sets the case as enabled or disabled

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

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

coroutine redbot.core.modlog.**get_next_case_number**(*guild: discord.guild.Guild*) → str
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 str

coroutine redbot.core.modlog.**get_case**(*case_number: int*, *guild: discord.guild.Guild*, *bot: redbot.core.bot.Red*) → 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

coroutine redbot.core.modlog.**get_all_cases**(*guild: discord.guild.Guild*, *bot: redbot.core.bot.Red*) → typing.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

coroutine redbot.core.modlog.**create_case**(*guild: discord.guild.Guild, created_at: datetime.datetime, action_type: str, user: typing.Union[discord.user.User, discord.member.Member], moderator: discord.member.Member = None, reason: str = None, until: datetime.datetime = None, channel: discord.channel.TextChannel = None*) → typing.Union[redbot.core.modlog.Case, NoneType]
Creates a new case

Parameters

- **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** (discord.User or discord.Member) – The user target by the action
- **moderator** (discord.Member) – The moderator who took the action

- **reason** (*str*) – The reason the action was taken
- **until** (*datetime*) – The time the action is in effect until
- **channel** (discord.TextChannel or discord.VoiceChannel) – The channel the action was taken in

Returns The newly created case

Return type *Case*

Raises RuntimeError – If the mod log channel doesn't exist

coroutine redbot.core.modlog.**get_casetype**(*name: str*, *guild: discord.guild.Guild = None*) → typing.Union[redbot.core.modlog.CaseType, NoneType]

Gets the case type

Parameters

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

Returns

Return type *CaseType* or None

coroutine redbot.core.modlog.**get_all_casetypes**(*guild: discord.guild.Guild = None*) → typing.List[redbot.core.modlog.CaseType]

Get all currently registered case types

Returns A list of case types

Return type list

coroutine redbot.core.modlog.**register_casetype**(*name: str*, *default_setting: bool*, *image: str*, *case_str: str*, *audit_type: str = 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

coroutine redbot.core.modlog.**register_casetypes**(*new_types: typing.List[dict]*) → typing.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

- RuntimeError
- KeyError
- ValueError
- AttributeError

See also:

redbot.core.modlog.register_casetype

coroutine redbot.core.modlog.**get_modlog_channel**(*guild: discord.guild.Guild*) → typing.Union[discord.channel.TextChannel, NoneType]

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 or None

Raises RuntimeError – If the modlog channel is not found

coroutine redbot.core.modlog.**set_modlog_channel**(*guild: discord.guild.Guild, channel: typing.Union[discord.channel.TextChannel, NoneType]*) → 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

coroutine redbot.core.modlog.**reset_cases**(*guild: discord.guild.Guild*) → bool

Wipes all modlog cases for the specified guild

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

Returns True if successful

Return type bool

CHAPTER 17

Command Invocation Context

The purpose of this module is to allow for Red to further customise the command invocation context provided by discord.py.

class redbot.core.**RedContext**(***attrs*)

Command invocation context for Red.

All context passed into commands will be of this type.

This class inherits from commands.Context.

coroutine send_help() → typing.List[discord.message.Message]

Send the command help message.

Returns A list of help messages which were sent to the user.

Return type list of discord.Message

coroutine send_interactive(*messages: typing.Iterable[str], box_lang: str = None, timeout: int = 15*) → typing.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.

coroutine tick() → bool

Add a tick reaction to the command message.

Returns True if adding the reaction succeeded.

Return type bool

CHAPTER 18

Utility Functions

18.1 Chat Formatting

redbot.core.utils.chat_formatting.**bold**(*text: str*) → 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**(*text1: typing.List[str], text2: typing.List[str]*) → str

Get two blocks of text in a borders.

Note: This will only work with a monospaced font.

Parameters

- **text1** (list of str) – The 1st block of text, with each string being a new line.
- **text2** (list of str) – The 2nd block of text. Should not be longer than text1.

Returns The bordered text.

Return type str

redbot.core.utils.chat_formatting.**box**(*text: str*, *lang: str = ''*) → 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*) → str
Get text prefixed with an error emoji.

Returns The new message.

Return type str

redbot.core.utils.chat_formatting.**escape**(*text: str*, *, *mass_mentions: bool = False*, *formatting: bool = 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.**info**(*text: str*) → str
Get text prefixed with an info emoji.

Returns The new message.

Return type str

redbot.core.utils.chat_formatting.**inline**(*text: str*) → 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*) → str
Get the given text in italics.

Parameters **text** (*str*) – The text to be marked up.

Returns The marked up text.

Return type str

redbot.core.utils.chat_formatting.**pagify**(*text: str, delims: typing.List[str] = ['\n'], *, priority: bool = False, escape_mass_mentions: bool = True, shorten_by: int = 8, page_length: int = 2000*) → typing.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** (list 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*) → str
Get text prefixed with a question emoji.

Returns The new message.

Return type str

redbot.core.utils.chat_formatting.**strikethrough**(*text: str*) → 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*) → 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*) → str
Get text prefixed with a warning emoji.

Returns The new message.

Return type str

18.2 Mod Helpers

redbot.core.utils.mod.**get_audit_reason**(*author: discord.member.Member*, *reason: str = 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 behidn the audit log action.

Returns The formatted audit log reason.

Return type str

coroutine redbot.core.utils.mod.**is_admin_or_superior**(*bot: redbot.core.bot.Red, obj: typing.Union[discord.message.Message, discord.member.Member, discord.role.Role]*)

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.

coroutine redbot.core.utils.mod.**is_mod_or_superior**(*bot: redbot.core.bot.Red, obj: typing.Union[discord.message.Message, discord.member.Member, discord.role.Role]*)

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.

coroutine redbot.core.utils.mod.**mass_purge**(*messages: typing.List[discord.message.Message], channel: discord.channel.TextChannel*)

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.

coroutine redbot.core.utils.mod.**slow_deletion**(*messages: typing.Iterable[discord.message.Message]*)

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: datetime.timedelta*)

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

CHAPTER 19

Indices and tables

- genindex
- modindex
- search

Python Module Index

r

Index

Symbols

A

B

C

D

E

F

G

H

I