Red API docs Documentation

Release 3

Various

Aug 06, 2017

Contents:

CHAPTER 1

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.

Basic Usage

```
from 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(config.foo())
```

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.

Config

class core.**Config**

You should always use *get_conf()* or *get_core_conf()* to initialize a Config object.

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:

```
conf.foo()
```

---

cog_name

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

unique_identifier

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

spawner

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

force_registration

A boolean that 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.

---

classmethod get_conf(*cog_instance*, *identifier*[, *force_registration=False*])

All cogs should use this classmethod to get a config instance.

Parameters

- **cog_instance** – The `self` arg inside of your cog's `__init__` function.
- **identifier** (*int*) – A random sequence of integers you provide to keep your cog's data safe in case of naming conflicts. This number should **never** change.
- **force_registration** (*Optional[bool]*) – See *force_registration*

classmethod get_core_conf(*identifier*[, *force_registration=False*])

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

Parameters

- **identifier** (*int*) – See *get_conf()*
- **force_registration** (*Optional[bool]*) – See *force_registration*

register_global(***kwargs*)

Registers default values for attributes you wish to store in *Config* at a global level.

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*)
Registers default values on a per-guild level. See *register_global()* for more details.

register_channel(***kwargs*)
Registers default values on a per-channel level. See *register_global()* for more details.

register_role(***kwargs*)
Registers default values on a per-role level. See *register_global()* for more details.

register_member(***kwargs*)
Registers default values on a per-member level. See *register_global()* for more details.

register_user(***kwargs*)
Registers default values on a per-user (independent of guild) level. See *register_global()* for more details.

guild(*guild*)
Returns a *Group* for the given guild.

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

channel(*channel*)
Returns a *Group* for the given channel. This does not currently support differences between text and voice channels.

Parameters **channel** (*discord.TextChannel*) – A discord.py text channel object.

role(*role*)
Returns a *Group* for the given role.

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

member(*member*)
Returns a *MemberGroup* for the given member.

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

user(*user*)
Returns a *Group* for the given user.

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

class core.config.**Group**
A "group" of data, 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
A dictionary of registered default values for this Group.

force_registration
See *Config.force_registration*.

__getattr__(*item*)

Takes in the next accessible item.

1. If it's found to be a group of data we return another Group object.
2. If it's found to be a data value we return a *Value* object.
3. If it is not found and *force_registration* is True then we raise AttributeError.
4. Otherwise return a *Value* object.

Parameters **item** (*str*) – The name of the item a cog is attempting to access through normal Python attribute access.

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

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

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

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

get_attr(*item*)
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.

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(self.conf.user(user).get_attr(item))
```

Parameters **item** (*str*) – The name of the data field in *Config*.

coroutine **set_attr**(*item*, *value*)
Please see *get_attr()* for more information.

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

---

all()
This method allows you to get "all" of a particular group of data. It will return the dictionary of all data for a particular Guild/Channel/Role/User/Member etc.

Return type dict

all_from_kind()
This method allows you to get all data from all entries in a given Kind. It will return a dictionary of Kind ID's -> data.

Return type dict

coroutine **clear**()
Wipes all data from the given Guild/Channel/Role/Member/User. If used on a global group, it will wipe all global data.

coroutine **clear_all**()
Wipes all data from all Guilds/Channels/Roles/Members/Users. If used on a global group, this method wipes all data from everything.

class core.config.**MemberGroup**
A specific group class for use with member data only. Inherits from *Group*. In this group data is stored as `GUILD_ID -> MEMBER_ID -> data`.

all_guilds()
Returns a dict of `MEMBER_ID -> data`.

Return type dict

class core.config.**Value**
A singular "value" of data.

identifiers
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
A reference to *Config.spawner*.

__call__(*[default=None]*)
Each `Value` object is created by the `__getattr__` method of a *Group* object and the data of the `Value` object is accessed by this method. It is a replacement for a `get()` method.

For example:

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

# Is equivalent to this

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

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

coroutine **set**(*value*)

Sets the value of the data element indicate by *identifiers*.

For 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)
```

Drivers

class core.config.drivers.red_base.**BaseDriver**

CHAPTER 2

Bank

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

Basic Usage

```
from 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
            )
        )
```

API Reference

Bank

The bank module.

class core.bank.**Account**

A single account. This class should ONLY be instantiated by the bank itself.

name

A string representing the name of the account

balance
An int representing the account's balance

created_at
A datetime.datetime object representing the time the account was created at

core.bank.**get_balance**(*member*)
Gets the current balance of a member.

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

Returns The balance for the specified member

Return type int

coroutine core.bank.**set_balance**(*member*, *amount*)
Sets the account balance for the member.

Parameters

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

Returns New account balance

Return type int

Raises **ValueError** – if amount is less than 0

coroutine core.bank.**withdraw_credits**(*member*, *amount*)
Removes a certain amount of credits from an account.

Parameters

- **member** (*discord.Member*) – A discord.py member object.
- **amount** (*int*) – The amount to withdraw

Returns New account balance

Return type int

Raises **ValueError** – if the amount is invalid or the account has insufficient funds

coroutine core.bank.**deposit_credits**(*member*, *amount*)
Adds a given amount of credits to an account.

Parameters

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

Returns New account balance

Return type int

Raises **ValueError** – if the amount is invalid

coroutine core.bank.**transfer_credits**(*from_*, *to*, *amount*)
Transfers 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 New balance for the account being transferred to

Return type int

Raises **ValueError** – if the amount is invalid or _from has insufficient funds

core.bank.**can_spend**(*member*, *amount*)
Determines if a member can spend the given amount.

Parameters

- **member** (*discord.Member*) – A discord.py member object.
- **amount** (*int*) – The amount the account wants to spend

Returns True if the account can spend the amount, otherwise False

Return type bool

coroutine core.bank.**wipe_bank**(*user*)
Deletes all accounts from the bank

Parameters **user** (*discord.User or discord.Member*) – a user to be used for wiping the bank

core.bank.**get_guild_accounts**(*guild*)
Gets all account data for the given guild

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

Returns List of account objects

Return type list(*Account*)

Raises **RuntimeError** – if the bank is currently global. Use *get_global_accounts()* to get all accounts if the bank is global

core.bank.**get_global_accounts**(*user*)
Gets all global account data

Parameters **user** (*discord.User*) – a discord.py user object

Returns List of account objects

Return type list(*Account*)

Raises **RuntimeError** – if the bank is currently guild specific. Use *get_guild_accounts()* to get all accounts for a guild if the bank is guild-specific

core.bank.**get_account**(*member*)
Gets the appropriate account for the given member.

Parameters **member** (*discord.User or discord.Member*) – the account to get

Returns The account for the specified member

Return type *Account*

core.bank.**is_global**()
Determines if the bank is currently global.

Returns True if bank is global, else False

Return type bool

coroutine core.bank.**set_global**(*global*, *user*)
Sets global status of the bank

Important: All accounts are reset when you switch!

Parameters

- **global** – True will set bank to global mode.
- **user** – Must be a Member object if changing TO global mode.

Returns New bank mode, True is global.

core.bank.**get_bank_name**(*guild*)
Gets the current bank name. If the bank is guild-specific the guild parameter is required.

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

Returns the bank name

Return type str

Raises **RuntimeError** – if guild parameter is missing and required

coroutine core.bank.**set_bank_name**(*name*, *guild*)
Sets the bank name. If the bank is guild-specific the guild parameter is required.

Parameters

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

Returns the new name for the bank

Return type str

Raises **RuntimeError** – if guild parameter is missing and required

core.bank.**get_currency_name**(*guild*)
Gets the currency name for the bank. If the bank is guild-specific the guild parameter is required.

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

Returns the currency name

Return type str

Raises **RuntimeError** – if guild parameter is missing and required

coroutine core.bank.**set_currency_name**(*name*, *guild*)
Sets the currency name for the bank. If the bank is guild-specific the guild parameter is required.

Parameters

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

Returns the new currency name for the bank

Return type str

Raises **RuntimeError** – if guild parameter is missing and required

core.bank.**get_default_balance**(*guild*)

Gets the current default balance amount. If the bank is guild-specific you must pass guild.

Parameters **guild** (*discord.Guild*) – The guild to get the default balance for

Returns the default balance

Return type int

Raises **RuntimeError** – if the guild parameter is missing and required

coroutine core.bank.**set_default_balance**(*amount*, *guild*)

Sets the default balance amount. If the bank is guild-specific you must pass guild.

Parameters

- **amount** (*int*) – The new amount for the default balance
- **guild** (*discord.Guild*) – The guild to set the default balance for

Returns the new default balance

Return type int

Raises

- **RuntimeError** – if the guild parameter is missing and required
- **ValueError** – if the new amount is invalid

CHAPTER 3

Data Manager

Data manager was introduced as a new feature in V3 as a way to move all of the bot's data out of the source code files. This was an important step towards making the bot pip installable.

Basic Usage

```
from core.data_manager import cog_data_path

class MyCog:
    def __init__(self):
        self.my_data_path = cog_data_path(self)
```

API Reference

core.data_manager.**load_basic_configuration**(*path*)

Loads the basic configuration necessary for config to work.

Parameters **path** (*pathlib.Path*) – Path pointing to the basic configuration file.

core.data_manager.**cog_data_path**([*cog_instance=None*])

This will return a path-like object to the directory to which a cog is allowed to write its own data.

Note: If cog instance is not provided it will return a path-like object pointing to the folder containing all cog data.

Parameters **cog_instance** (*Optional*) – An instantiated cog object.

Returns A path-like object.

Return type pathlib.Path

core.data_manager.**core_data_path**()

This will return a path-like object to the directory to which core cogs are allowed to write data.

Returns A path-like object.

Return type pathlib.Path

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

Symbols

A

B

C

D

F

G

I

L

M

N

R

S

T

U

W