SuperHelper Documentation

Contents

Module SuperHelper

Sub-modules

- SuperHelper.Core
- SuperHelper.Modules

Variables

Variable AppDir

Path to the application directory.

Variable AppName

Name of the application.

Module SuperHelper.Core

Sub-modules

- SuperHelper.Core.Config
- SuperHelper.Core.Utils
- SuperHelper.Core.core_cli
- SuperHelper.Core.core_commands
- SuperHelper.Core.core_loader

Functions

Function load_added_modules

```
def load_added_modules(
    config: dict
) -> list
```

Loads all added modules.

Returns —-= A list of a 2-tuple elements, where the first index is the click.command object, and the second index is the technical name of the command. For example:

```
[(main, "main"), ...]
```

The first index can be added to a click.group, i.e the cli function.

Function load_core_commands

```
def load_core_commands() -> list
```

Loads the Core CLI commands.

Returns —-= A list of a 2-tuple elements, where the first index is the click.command object, and the second index is the technical name of the command. For example:

```
[(add_modules, "core_add"), ...]
```

The first index can be added to a click.group, i.e the cli function.

Function `main_entry`

```
def main_entry() -> NoReturn
```

Function `pass_config`

```
def pass_config(
    core: bool = None,
    module_name: str = None,
    lock: bool = False,
    param_name: str = 'config'
) -> Callable
```

Passes the requested config to decorated functions.

The wrapped function will receive the config (as requested). When the function returns (or raises SystemExit), this decorator will capture that signal, save the config (if locked) before returning (or re-raising SystemExit).

Args —-= `core` : bool : Whether to request core config.

`module_name` : str The name of the module.
`lock` : bool Whether to lock the config, i.e allow writing to the config.
`param_name` : str The name of the parameter that the config will be passed as.

Returns —-= A Callable instance (the decorated function).

Raises —-= SystemExit : Re-raises the SystemExit() raised by the wrapped function.

ValueError Both core and module_name are specified.

Function `run_startup`

```
def run_startup()
```

Function `save_config`

```
def save_config()
```

Saves application config.

Module `SuperHelper.Core.Config`

Sub-modules

- SuperHelper.Core.Config.app_config
- SuperHelper.Core.Config.config_class

Functions

Function `load_app_config`

```
def load_app_config(
    config_path: ~PathLike
) -> NoneType
```

Loads the configuration of the application.

Args —-= `config_path` : PathLike : The path to config file.

Returns —-= None

Raises —-= SystemExit : Config file is unreadable.

Function `make_config_global`

```
def make_config_global(
    cfg: Config
) -> NoneType
```

Makes the configuration global.

Args —-= `cfg` : Config : The Config instance.

Returns —-= None

Function `pass_config`

```
def pass_config(
    core: bool = None,
    module_name: str = None,
    lock: bool = False,
    param_name: str = 'config'
) -> Callable
```

Passes the requested config to decorated functions.

The wrapped function will receive the config (as requested). When the function returns (or raises SystemExit), this decorator will capture that signal, save the config (if locked) before returning (or re-raising SystemExit).

Args —-= `core` : bool : Whether to request core config.

`module_name` : str The name of the module.
`lock` : bool Whether to lock the config, i.e allow writing to the config.
`param_name` : str The name of the parameter that the config will be passed as.

Returns —-= A Callable instance (the decorated function).

Raises —-= SystemExit : Re-raises the SystemExit() raised by the wrapped function.

ValueError Both core and module_name are specified.

Function `save_app_config`

```
def save_app_config(
    config: SuperHelper.Core.Config.config_class.Config,
    config_path: ~PathLike
) -> NoneType
```

Saves the configuration of the application.

Args —-= `config` : Config : The global Config instance

`config_path` : PathLike The path to config file

Returns —-= None

Raises —-= SystemExit : Config file is not writable.

Classes

Class `Config`

```
class Config(
    core: dict[str, ...] = None,
    modules: dict[str, dict[str, ...]] = None
)
```

The configuration of the application.

Static methods

`Method from_dict`

```
def from_dict(
    config: dict[str]
) -> SuperHelper.Core.Config.config_class.Config
```

Methods

Method `apply_core_patch`

```
def apply_core_patch(
    self,
    config: dict[str, ...]
) -> NoneType
```

Applies a new patch to core configuration.

This function should only be used by Core CLI.

Args —-= `config` : dict[str, ...] : The patch of the configuration.

Returns —-= None

Raises —-= RuntimeError : An error has occurred in self.get_core_config()

Method `apply_module_patch`

```
def apply_module_patch(
    self,
    module_name: str,
    config: dict[str, ...]
) -> NoneType
```

Applies a new patch to the module configuration.

Args —-= `module_name` : str : The name of the module to apply patch to.

`config` : dict[str, ...] The patch of the configuration.

Returns —-= None

Method `get_core_config`

```
def get_core_config(
    self,
    lock: bool = True
) -> dict
```

Gets the configuration of Core CLI.

This function is intended for internal use only, used for the decorator pass_config().

Args —-= `lock` : bool : Whether to lock the config or not.

Returns —-= A dictionary mapping keys to corresponding values of the core config. Each entry is represented by a key-value pair of the dictionary. For example:

```
{"DEBUG": ..., "INSTALLED_MODULES": [...]}
```

The keys are always strings, and the values can be of any JSON-serializable type.

Raises —-= RuntimeError : The core config is locked by another call.

Method **`get_module_config`**

```
def get_module_config(
    self,
    module_name: str,
    lock: bool = True
) -> dict
```

Gets the configuration of the specified module.

This function is intended for internal use only, used for the decorator pass_config().

Args —-= **`module_name`** : str : The name of the module that the config belongs to.

`lock` : bool Whether to lock the config or not.

Returns —-= A dictionary mapping keys to corresponding values of the module config. Each entry is represented by a key-value pair of the dictionary. For example:

```
{"DEBUG": ..., "INSTALLED_MODULES": [...]}
```

The keys are always strings, and the values can be of any JSON-serializable type.

Raises —-= RuntimeError : The module config is locked by another call.

Method **`set_core_config`**

```
def set_core_config(
    self,
    config: dict[str, ...]
) -> NoneType
```

Sets the configuration of Core CLI.

This function is intended for internal use only, used for the decorator pass_config().

Args —-= **`config`** : dict[str, ...] : A dictionary with string keys of the core configuration.

Returns —-= None

Raises —-= RuntimeError : The last retrieval of the core config was not locked, hence it is read-only.

Method **`set_module_config`**

```
def set_module_config(
    self,
    module_name: str,
    config: dict[str, ...]
) -> NoneType
```

Sets the module configuration.

This function is intended for internal use only, used for the decorator pass_config().

Args —-= `module_name` : str : The name of the module that the config belongs to.

`config` : dict[str, ...] A dictionary with string keys of the core configuration.

Returns —-= None

Raises —-= RuntimeError : The last retrieval of the module config was not locked, hence it is read-only.

Module `SuperHelper.Core.Config.app_config`

Functions

Function `load_app_config`

```
def load_app_config(
    config_path: ~PathLike
) -> NoneType
```

Loads the configuration of the application.

Args —-= `config_path` : PathLike : The path to config file.

Returns —-= None

Raises —-= SystemExit : Config file is unreadable.

Function `save_app_config`

```
def save_app_config(
    config: SuperHelper.Core.Config.config_class.Config,
    config_path: ~PathLike
) -> NoneType
```

Saves the configuration of the application.

Args —-= `config` : Config : The global Config instance

`config_path` : PathLike The path to config file

Returns —-= None

Raises —-= SystemExit : Config file is not writable.

Module `SuperHelper.Core.Config.config_class`

Functions

Function `make_config_global`

```
def make_config_global(
    cfg: Config
) -> NoneType
```

Makes the configuration global.

Args —-= `cfg` : Config : The Config instance.

Returns —-= None

Function `pass_config`

```
def pass_config(
    core: bool = None,
    module_name: str = None,
    lock: bool = False,
    param_name: str = 'config'
) -> Callable
```

Passes the requested config to decorated functions.

The wrapped function will receive the config (as requested). When the function returns (or raises SystemExit), this decorator will capture that signal, save the config (if locked) before returning (or re-raising SystemExit).

Args —-= `core` : bool : Whether to request core config.

`module_name` : str The name of the module.
`lock` : bool Whether to lock the config, i.e allow writing to the config.
`param_name` : str The name of the parameter that the config will be passed as.

Returns —-= A Callable instance (the decorated function).

Raises —-= SystemExit : Re-raises the SystemExit() raised by the wrapped function.

ValueError Both core and module_name are specified.

Classes

Class `Config`

```
class Config(
    core: dict[str, ...] = None,
    modules: dict[str, dict[str, ...]] = None
)
```

The configuration of the application.

Static methods

`Method from_dict`

```
def from_dict(
    config: dict[str]
) -> SuperHelper.Core.Config.config_class.Config
```

Methods

Method `apply_core_patch`

```
def apply_core_patch(
    self,
    config: dict[str, ...]
) -> NoneType
```

Applies a new patch to core configuration.

This function should only be used by Core CLI.

Args —-= `config` : dict[str, ...] : The patch of the configuration.

Returns —-= None

Raises —-= RuntimeError : An error has occurred in self.get_core_config()

Method `apply_module_patch`

```
def apply_module_patch(
    self,
    module_name: str,
    config: dict[str, ...]
) -> NoneType
```

Applies a new patch to the module configuration.

Args —-= `module_name` : str : The name of the module to apply patch to.

`config` : dict[str, ...] The patch of the configuration.

Returns —-= None

Method `get_core_config`

```
def get_core_config(
    self,
    lock: bool = True
) -> dict
```

Gets the configuration of Core CLI.

This function is intended for internal use only, used for the decorator pass_config().

Args —-= `lock` : bool : Whether to lock the config or not.

Returns —-= A dictionary mapping keys to corresponding values of the core config. Each entry is represented by a key-value pair of the dictionary. For example:

```
{"DEBUG": ..., "INSTALLED_MODULES": [...]}
```

The keys are always strings, and the values can be of any JSON-serializable type.

Raises —-= RuntimeError : The core config is locked by another call.

Method `get_module_config`

```
def get_module_config(
    self,
    module_name: str,
    lock: bool = True
) -> dict
```

Gets the configuration of the specified module.

This function is intended for internal use only, used for the decorator pass_config().

Args —-= `module_name` : str : The name of the module that the config belongs to.

`lock` : bool Whether to lock the config or not.

Returns —-= A dictionary mapping keys to corresponding values of the module config. Each entry is represented by a key-value pair of the dictionary. For example:

```
{"DEBUG": ..., "INSTALLED_MODULES": [...]}
```

The keys are always strings, and the values can be of any JSON-serializable type.

Raises —-= RuntimeError : The module config is locked by another call.

Method `set_core_config`

```
def set_core_config(
    self,
    config: dict[str, ...]
) -> NoneType
```

Sets the configuration of Core CLI.

This function is intended for internal use only, used for the decorator pass_config().

Args —-= `config` : dict[str, ...] : A dictionary with string keys of the core configuration.

Returns —-= None

Raises —-= RuntimeError : The last retrieval of the core config was not locked, hence it is read-only.

Method `set_module_config`

```
def set_module_config(
    self,
    module_name: str,
    config: dict[str, ...]
) -> NoneType
```

Sets the module configuration.

This function is intended for internal use only, used for the decorator pass_config().

Args —-= `module_name` : str : The name of the module that the config belongs to.

`config` : dict[str, ...] A dictionary with string keys of the core configuration.

Returns —-= None

Raises —-= RuntimeError : The last retrieval of the module config was not locked, hence it is read-only.

Module `SuperHelper.Core.Utils`

Sub-modules

- SuperHelper.Core.Utils.bit_ops
- SuperHelper.Core.Utils.crypto_ops
- SuperHelper.Core.Utils.file_ops
- SuperHelper.Core.Utils.logger
- SuperHelper.Core.Utils.type_ensure
- SuperHelper.Core.Utils.type_hinting

Functions

Function `setup_core_logger`

```
def setup_core_logger(
    logging_path: ~PathLike
) -> logging.Logger
```

Sets up the core logger.

Args —-= `logging_path` : PathLike : The path to the logging file.

Returns —-= A logging.Logger instance with name set to SuperHelper.

Classes

Class `BitOps`

```
class BitOps
```

A utility class for bitwise operations.

Static methods

`Method is_bit_set`

```
def is_bit_set(
    i: int,
    pos: int
) -> bool
```

Checks if the pos-th bit of the integer i is set.

Args —-= `i` : int : The integer to check.

`pos` : int The zero-indexed position of the bit (from LSB) to check.

Returns —-= True if the specified bit is set, otherwise False

`Method set_bit`

```
def set_bit(
    i: int,
    pos: int
) -> int
```

Sets the the pos-th bit of the integer i.

Args —-= `i` : int : The integer to modify.

`pos` : int The zero-indexed position of the bit (from LSB) to set.

Returns —-= The integer with the specified bit set.

`Method unset_bit`

```
def unset_bit(
    i: int,
    pos: int
) -> int
```

Unsets the the pos-th bit of the integer i.

Args —-= `i` : int : The integer to modify.

`pos` : int The zero-indexed position of the bit (from LSB) to unset.

Returns —-= The integer with the specified bit unset.

Class `Cryptographer`

```
class Cryptographer(
    salt: bytes,
    auth_key: bytes,
    encrypt: bool = True
)
```

A utility class for cryptographic functions.

Initialises a Cryptographer instance.

Args —-= `salt` : bytes : The raw salt, in bytes.

`auth_key` : bytes The authentication key, in bytes.
`encrypt` : bool True to make an encrypter, otherwise False.

Static methods

`Method decode_salt`

```
def decode_salt(
    salt: str
) -> bytes
```

Decodes the salt string to raw salt.

Args —-= `salt` : str : The Base64-encoded string of the raw salt.

Returns —-= The raw salt

`Method encode_salt`

```
def encode_salt(
    salt: bytes
) -> str
```

Encodes the raw salt as string.

Args —-= `salt` : bytes : The raw salt, in bytes.

Returns —-= The Base64-encoded string of the raw salt

`Method make_decrypter`

```
def make_decrypter(
    salt: str,
    key: str
) -> SuperHelper.Core.Utils.crypto_ops.Cryptographer
```

Makes a Fernet decrypter for salt and key.

Args —-= `salt` : str : The Base64-encoded string of the raw salt.

`key` : str The authentication key.

Returns —-= A Cryptographer instance, which can be used to decrypt data.

`Method make_encrypter`

```
def make_encrypter(
    salt: str,
    key: str
) -> SuperHelper.Core.Utils.crypto_ops.Cryptographer
```

Makes a Fernet encrypter for salt and key.

Args —-= `salt` : str : The Base64-encoded string of the raw salt.

`key` : str The authentication key.

Returns —-= A Cryptographer instance, which can be used to encrypt data.

`Method make_fernet`

```
def make_fernet(
    key: bytes
) -> cryptography.fernet.Fernet
```

Makes a Fernet encrypter/decrypter from the derived key.

Args —-= `key` : bytes : The derived key, in bytes.

Returns —-= A Fernet instance, which can be used to either encrypt or decrypt data.

`Method make_kdf`

```
def make_kdf(
    salt: bytes
) -> cryptography.hazmat.primitives.kdf.pbkdf2.PBKDF2HMAC
```

Makes a key derivation function from raw salt.

Args —-= `salt` : bytes : The raw salt, in bytes.

Returns —-= A PBKDF2HMAC instance, which can be used to derive key from the authentication key.

`Method make_salt`

```
def make_salt() -> bytes
```

Generates a cryptographically secure salt for cryptography.

Returns —-= A 16-byte raw salt

Methods

Method `decrypt`

```
def decrypt(
    self,
    encrypted_data: bytes
) -> bytes
```

Decrypts the encrypted data.

Args —-= `encrypted_data` : bytes : The encrypted data to be decrypted.

Returns —-= The decrypted data, in bytes, which is decrypted using the Fernet (created by Cryptography.make_fernet)

Method `encrypt`

```
def encrypt(
    self,
    raw_data: bytes
) -> bytes
```

Encrypts raw data.

Args —-= `raw_data` : bytes : The raw data to be encrypted.

Returns —-= The encrypted data, in bytes, which is encrypted using the Fernet (created by Cryptography.make_fernet)

Raises —-= ValueError : A decrypter is used to encrypt.

Method `get_salt_string`

```
def get_salt_string(
    self
) -> str
```

String-ify the raw salt.

Returns —-= The Base64-encoded string of the raw salt.

Class `FP`

```
class FP(
    value,
    names=None,
    *,
    module=None,
    qualname=None,
    type=None,
    start=1
)
```

Contains file permission flags.

R = Read

W = Write

X = Execute

USR = User (file owner)

GRP = Group owner

OTH = Other users/groups

Ancestors (in MRO)

- enum.Flag
- enum.Enum

Class variables

Variable `R_GRP` Group readable.

Variable `R_OTH` Other readable.

Variable `R_USR` User readable.

Variable `W_GRP` Group writable.

Variable `W_OTH` Other writable.

Variable `W_USR` User writable.

Variable `X_GRP` Group executable.

Variable `X_OTH` Other executable.

Variable `X_USR` User executable.

Class `FileOps`

```
class FileOps
```

A utility class for file ownership and permissions.

Static methods

`Method check_fp`

```
def check_fp(
    path: ~PathLike,
    fp: SuperHelper.Core.Utils.file_ops.FP
) -> bool
```

Checks if the file contains the specified file permissions.

:param path: Path to the file to check :type path: PathLike :param fp: The flags of the file permissions to check. :type fp: FP :return: True if all the flags are valid, otherwise False :rtype: bool

`Method get_stat`

```
def get_stat(
    path: ~PathLike
) -> os.stat_result
```

Gets the stat of file pointed by the path.

This function is decorated by @cache to reduce the amount of syscall, since os.stat is an expensive function.

Args —-= `path` : PathLike : Path to the file to check

Returns —-= An os.stat_result instance containing the stat of the file.

`Method is_group_executable`

```
def is_group_executable(
    path: ~PathLike
) -> bool
```

Checks if the group owner of the file can execute it.

:param path: Path to the file to check :type path: PathLike :return: True if the file is executable by its group owner, otherwise False :rtype: bool

`Method is_group_readable`

```
def is_group_readable(
    path: ~PathLike
) -> bool
```

Checks if the group owner of the file can read it.

:param path: Path to the file to check :type path: PathLike :return: True if the file is readable by its group owner, otherwise False :rtype: bool

`Method is_group_writable`

```
def is_group_writable(
    path: ~PathLike
) -> bool
```

Checks if the group owner of the file can write to it.

:param path: Path to the file to check :type path: PathLike :return: True if the file is writable by its group owner, otherwise False :rtype: bool

`Method is_mine`

```
def is_mine(
    path: ~PathLike
) -> bool
```

Checks if the file is owned by the current user.

:param path: Path to the file to check :type path: PathLike :return: True if the file is owned by the current user, otherwise False :rtype: bool

`Method is_other_executable`

```
def is_other_executable(
    path: ~PathLike
) -> bool
```

Checks if the other users or groups can execute the file.

:param path: Path to the file to check :type path: PathLike :return: True if the file is executable by them, otherwise False :rtype: bool

`Method is_other_readable`

```
def is_other_readable(
    path: ~PathLike
) -> bool
```

Checks if the other users or groups can read the file.

:param path: Path to the file to check :type path: PathLike :return: True if the file is readable by them, otherwise False :rtype: bool

`Method is_other_writable`

```
def is_other_writable(
    path: ~PathLike
) -> bool
```

Checks if the other users or groups can write the file.

:param path: Path to the file to check :type path: PathLike :return: True if the file is writable by them, otherwise False :rtype: bool

`Method is_owner_executable`

```
def is_owner_executable(
    path: ~PathLike
) -> bool
```

Checks if the owner of the file can execute it.

:param path: Path to the file to check :type path: PathLike :return: True if the file is executable by its owner, otherwise False :rtype: bool

Method `is_owner_readable`

```
def is_owner_readable(
    path: ~PathLike
) -> bool
```

Checks if the owner of the file can read it.

:param path: Path to the file to check :type path: PathLike :return: True if the file is readable by its owner, otherwise False :rtype: bool

Method `is_owner_writable`

```
def is_owner_writable(
    path: ~PathLike
) -> bool
```

Checks if the owner of the file can write to it.

:param path: Path to the file to check :type path: PathLike :return: True if the file is writable by its owner, otherwise False :rtype: bool

Method `is_roots`

```
def is_roots(
    path: ~PathLike
) -> bool
```

Check if the file is owned by root.

:param path: Path to the file to check :type path: PathLike :return: True if the file is owned by root, otherwise False :rtype: bool

Method `is_user_own`

```
def is_user_own(
    uid: int,
    path: ~PathLike
) -> bool
```

Checks if the file is owned by the user with uid.

:param uid: The UID of the user :type uid: int :param path: Path to the file to check :type path: PathLike :return: True if the file is owned by the uid, otherwise False :rtype: bool

Class `TypeCheck`

```
class TypeCheck
```

A utility class for type checking functions.

Static methods

Method `ensure_bool`

```
def ensure_bool(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type bool.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_bytearray

```
def ensure_bytearray(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type bytearray.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_bytes

```
def ensure_bytes(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type bytes.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_complex

```
def ensure_complex(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type complex.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_custom

```
def ensure_custom(
    t: type,
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of the expected type.

Args —-= `t` : type : The expected type of the object.

`obj` : object The object to check.
`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

`Method ensure_dict`

```
def ensure_dict(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type dict.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

`Method ensure_float`

```
def ensure_float(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type float.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

`Method ensure_frozenset`

```
def ensure_frozenset(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type frozenset.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

`Method ensure_function`

```
def ensure_function(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is a function.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_generator

```
def ensure_generator(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is a generator.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_int

```
def ensure_int(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type int.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_list

```
def ensure_list(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type list.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_memoryview

```
def ensure_memoryview(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type memoryview.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_path_like

```
def ensure_path_like(
    obj: Ellipsis,
    name: str = None
)
```

Ensures the object can be used as a path.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_set

```
def ensure_set(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type set.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_str

```
def ensure_str(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type str.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_tuple

```
def ensure_tuple(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type tuple.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Module `SuperHelper.Core.Utils.bit_ops`

Classes

Class `BitOps`

```
class BitOps
```

A utility class for bitwise operations.

Static methods

`Method is_bit_set`

```
def is_bit_set(
    i: int,
    pos: int
) -> bool
```

Checks if the pos-th bit of the integer i is set.

Args —-= `i` : int : The integer to check.

`pos` : int The zero-indexed position of the bit (from LSB) to check.

Returns —-= True if the specified bit is set, otherwise False

`Method set_bit`

```
def set_bit(
    i: int,
    pos: int
) -> int
```

Sets the the pos-th bit of the integer i.

Args —-= `i` : int : The integer to modify.

`pos` : int The zero-indexed position of the bit (from LSB) to set.

Returns —-= The integer with the specified bit set.

`Method unset_bit`

```
def unset_bit(
    i: int,
    pos: int
) -> int
```

Unsets the the pos-th bit of the integer i.

Args —-= `i` : int : The integer to modify.

`pos` : int The zero-indexed position of the bit (from LSB) to unset.

Returns —-= The integer with the specified bit unset.

Module `SuperHelper.Core.Utils.crypto_ops`

Classes

Class `Cryptographer`

```
class Cryptographer(
    salt: bytes,
    auth_key: bytes,
    encrypt: bool = True
)
```

A utility class for cryptographic functions.

Initialises a Cryptographer instance.

Args —-= `salt` : bytes : The raw salt, in bytes.

`auth_key` : bytes The authentication key, in bytes.
`encrypt` : bool True to make an encrypter, otherwise False.

Static methods

`Method decode_salt`

```
def decode_salt(
    salt: str
) -> bytes
```

Decodes the salt string to raw salt.

Args —-= `salt` : str : The Base64-encoded string of the raw salt.

Returns —-= The raw salt

`Method encode_salt`

```
def encode_salt(
    salt: bytes
) -> str
```

Encodes the raw salt as string.

Args —-= `salt` : bytes : The raw salt, in bytes.

Returns —-= The Base64-encoded string of the raw salt

`Method make_decrypter`

```
def make_decrypter(
    salt: str,
    key: str
) -> SuperHelper.Core.Utils.crypto_ops.Cryptographer
```

Makes a Fernet decrypter for salt and key.

Args —-= `salt` : str : The Base64-encoded string of the raw salt.

`key` : str The authentication key.

Returns —-= A Cryptographer instance, which can be used to decrypt data.

Method `make_encrypter`

```
def make_encrypter(
    salt: str,
    key: str
) -> SuperHelper.Core.Utils.crypto_ops.Cryptographer
```

Makes a Fernet encrypter for salt and key.

Args —-= `salt` : str : The Base64-encoded string of the raw salt.

`key` : str The authentication key.

Returns —-= A Cryptographer instance, which can be used to encrypt data.

Method `make_fernet`

```
def make_fernet(
    key: bytes
) -> cryptography.fernet.Fernet
```

Makes a Fernet encrypter/decrypter from the derived key.

Args —-= `key` : bytes : The derived key, in bytes.

Returns —-= A Fernet instance, which can be used to either encrypt or decrypt data.

Method `make_kdf`

```
def make_kdf(
    salt: bytes
) -> cryptography.hazmat.primitives.kdf.pbkdf2.PBKDF2HMAC
```

Makes a key derivation function from raw salt.

Args —-= `salt` : bytes : The raw salt, in bytes.

Returns —-= A PBKDF2HMAC instance, which can be used to derive key from the authentication key.

Method `make_salt`

```
def make_salt() -> bytes
```

Generates a cryptographically secure salt for cryptography.

Returns —-= A 16-byte raw salt

Methods

Method `decrypt`

```
def decrypt(
    self,
    encrypted_data: bytes
) -> bytes
```

Decrypts the encrypted data.

Args —-= `encrypted_data` : bytes : The encrypted data to be decrypted.

Returns —-= The decrypted data, in bytes, which is decrypted using the Fernet (created by Cryptography.make_fernet)

Method `encrypt`

```
def encrypt(
    self,
    raw_data: bytes
) -> bytes
```

Encrypts raw data.

Args —-= `raw_data` : bytes : The raw data to be encrypted.

Returns —-= The encrypted data, in bytes, which is encrypted using the Fernet (created by Cryptography.make_fernet)

Raises —-= ValueError : A decrypter is used to encrypt.

Method `get_salt_string`

```
def get_salt_string(
    self
) -> str
```

String-ify the raw salt.

Returns —-= The Base64-encoded string of the raw salt.

Module `SuperHelper.Core.Utils.file_ops`

Classes

Class `FP`

```
class FP(
    value,
    names=None,
    *,
    module=None,
    qualname=None,
    type=None,
    start=1
)
```

Contains file permission flags.

R = Read

W = Write

X = Execute

USR = User (file owner)

GRP = Group owner

OTH = Other users/groups

Ancestors (in MRO)

- enum.Flag
- enum.Enum

Class variables

Variable `R_GRP` Group readable.

Variable `R_OTH` Other readable.

Variable `R_USR` User readable.

Variable `W_GRP` Group writable.

Variable `W_OTH` Other writable.

Variable `W_USR` User writable.

Variable `X_GRP` Group executable.

Variable `X_OTH` Other executable.

Variable `X_USR` User executable.

Class `FileOps`

```
class FileOps
```

A utility class for file ownership and permissions.

Static methods

`Method check_fp`

```
def check_fp(
    path: ~PathLike,
    fp: SuperHelper.Core.Utils.file_ops.FP
) -> bool
```

Checks if the file contains the specified file permissions.

:param path: Path to the file to check :type path: PathLike :param fp: The flags of the file permissions to check. :type fp: FP :return: True if all the flags are valid, otherwise False :rtype: bool

`Method get_stat`

```
def get_stat(
    path: ~PathLike
) -> os.stat_result
```

Gets the stat of file pointed by the path.

This function is decorated by @cache to reduce the amount of syscall, since os.stat is an expensive function.

Args —-= `path` : PathLike : Path to the file to check

Returns —-= An os.stat_result instance containing the stat of the file.

`Method is_group_executable`

```
def is_group_executable(
    path: ~PathLike
) -> bool
```

Checks if the group owner of the file can execute it.

:param path: Path to the file to check :type path: PathLike :return: True if the file is executable by its group owner, otherwise False :rtype: bool

`Method is_group_readable`

```
def is_group_readable(
    path: ~PathLike
) -> bool
```

Checks if the group owner of the file can read it.

:param path: Path to the file to check :type path: PathLike :return: True if the file is readable by its group owner, otherwise False :rtype: bool

`Method is_group_writable`

```
def is_group_writable(
    path: ~PathLike
) -> bool
```

Checks if the group owner of the file can write to it.

:param path: Path to the file to check :type path: PathLike :return: True if the file is writable by its group owner, otherwise False :rtype: bool

`Method is_mine`

```
def is_mine(
    path: ~PathLike
) -> bool
```

Checks if the file is owned by the current user.

:param path: Path to the file to check :type path: PathLike :return: True if the file is owned by the current user, otherwise False :rtype: bool

`Method is_other_executable`

```
def is_other_executable(
    path: ~PathLike
) -> bool
```

Checks if the other users or groups can execute the file.

:param path: Path to the file to check :type path: PathLike :return: True if the file is executable by them, otherwise False :rtype: bool

`Method is_other_readable`

```
def is_other_readable(
    path: ~PathLike
) -> bool
```

Checks if the other users or groups can read the file.

:param path: Path to the file to check :type path: PathLike :return: True if the file is readable by them, otherwise False :rtype: bool

`Method is_other_writable`

```
def is_other_writable(
    path: ~PathLike
) -> bool
```

Checks if the other users or groups can write the file.

:param path: Path to the file to check :type path: PathLike :return: True if the file is writable by them, otherwise False :rtype: bool

`Method is_owner_executable`

```
def is_owner_executable(
    path: ~PathLike
) -> bool
```

Checks if the owner of the file can execute it.

:param path: Path to the file to check :type path: PathLike :return: True if the file is executable by its owner, otherwise False :rtype: bool

`Method is_owner_readable`

```
def is_owner_readable(
    path: ~PathLike
) -> bool
```

Checks if the owner of the file can read it.

:param path: Path to the file to check :type path: PathLike :return: True if the file is readable by its owner, otherwise False :rtype: bool

`Method is_owner_writable`

```
def is_owner_writable(
    path: ~PathLike
) -> bool
```

Checks if the owner of the file can write to it.

:param path: Path to the file to check :type path: PathLike :return: True if the file is writable by its owner, otherwise False :rtype: bool

`Method is_roots`

```
def is_roots(
    path: ~PathLike
) -> bool
```

Check if the file is owned by root.

:param path: Path to the file to check :type path: PathLike :return: True if the file is owned by root, otherwise False :rtype: bool

`Method is_user_own`

```
def is_user_own(
    uid: int,
    path: ~PathLike
) -> bool
```

Checks if the file is owned by the user with uid.

:param uid: The UID of the user :type uid: int :param path: Path to the file to check :type path: PathLike :return: True if the file is owned by the uid, otherwise False :rtype: bool

Module `SuperHelper.Core.Utils.logger`

Functions

Function `setup_core_logger`

```
def setup_core_logger(
    logging_path: ~PathLike
) -> logging.Logger
```

Sets up the core logger.

Args —-= `logging_path` : PathLike : The path to the logging file.

Returns —-= A logging.Logger instance with name set to SuperHelper.

Module `SuperHelper.Core.Utils.type_ensure`

Classes

Class `TypeCheck`

```
class TypeCheck
```

A utility class for type checking functions.

Static methods

`Method ensure_bool`

```
def ensure_bool(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type bool.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

`Method ensure_bytearray`

```
def ensure_bytearray(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type bytearray.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

`Method ensure_bytes`

```
def ensure_bytes(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type bytes.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

`Method ensure_complex`

```
def ensure_complex(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type complex.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

`Method ensure_custom`

```
def ensure_custom(
    t: type,
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of the expected type.

Args —-= `t` : type : The expected type of the object.

`obj` : object The object to check.
`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

`Method ensure_dict`

```
def ensure_dict(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type dict.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_float

```
def ensure_float(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type float.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_frozenset

```
def ensure_frozenset(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type frozenset.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_function

```
def ensure_function(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is a function.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_generator

```
def ensure_generator(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is a generator.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_int

```
def ensure_int(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type int.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_list

```
def ensure_list(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type list.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_memoryview

```
def ensure_memoryview(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type memoryview.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_path_like

```
def ensure_path_like(
    obj: Ellipsis,
    name: str = None
)
```

Ensures the object can be used as a path.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_set

```
def ensure_set(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type set.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_str

```
def ensure_str(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type str.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Method ensure_tuple

```
def ensure_tuple(
    obj: Ellipsis,
    name: str = None
) -> NoneType
```

Ensures the object is of type tuple.

Args —-= `obj` : object : The object to check.

`name` : str The name of the object.

Returns —-= None

Raises —-= TypeError : The type of the object is not the specified type.

Module `SuperHelper.Core.Utils.type_hinting`

Variables

Variable `PathLike`

Type: `type`

PathLike objects can be used as a path. It can be of type str, bytes or os.PathLike.

Module SuperHelper.Core.core_cli

Functions

Function `load_config`

```
def load_config()
```

Loads application config.

Function `main_entry`

```
def main_entry() -> NoReturn
```

Function `run_startup`

```
def run_startup()
```

Function `save_config`

```
def save_config()
```

Saves application config.

Module SuperHelper.Core.core_commands

Functions

Function `load_core_commands`

```
def load_core_commands() -> list
```

Loads the Core CLI commands.

Returns —-= A list of a 2-tuple elements, where the first index is the click.command object, and the second index is the technical name of the command. For example:

```
[(add_modules, "core_add"), ...]
```

The first index can be added to a click.group, i.e the cli function.

Module SuperHelper.Core.core_loader

Functions

Function `load_added_modules`

```
def load_added_modules(
    config: dict
) -> list
```

Loads all added modules.

Returns —-= A list of a 2-tuple elements, where the first index is the click.command object, and the second index is the technical name of the command. For example:

```
[(main, "main"), ...]
```

The first index can be added to a click.group, i.e the cli function.

Module `SuperHelper.Modules`

Sub-modules

- SuperHelper.Modules.FocusEnabler
- SuperHelper.Modules.Stenographer

Module `SuperHelper.Modules.FocusEnabler`

Module `SuperHelper.Modules.Stenographer`
