precept

Release 0.6.7

Philippe Duval

Jan 15, 2022

CONTENTS

1 Usage **1**
- 1.1 Install . . . 1
- 1.2 Write async console applications . . . 1
  - 1.2.1 Starting the application . . . 1
- 1.3 Configs . . . 2
  - 1.3.1 Example . . . 2
  - 1.3.2 Config file . . . 2
  - 1.3.3 Config format . . . 3

2 Concepts **5**
- 2.1 Events . . . 5
  - 2.1.1 cli events . . . 5
- 2.2 Services . . . 6
  - 2.2.1 Service events . . . 6
- 2.3 Plugins . . . 6

3 Application list **7**

4 precept **9**
- 4.1 precept package . . . 9
  - 4.1.1 Errors . . . 15
  - 4.1.2 Subpackages . . . 15
    - 4.1.2.1 precept.console . . . 15
    - 4.1.2.2 precept.events . . . 17

Python Module Index **19**

Index **21**

CHAPTER

ONE

USAGE

1.1 Install

Install with pip:

```
pip install precept
```

1.2 Write async console applications

Precept comes with many classes and functions to build async applications, the main class consist of *Precept* which you subclass to create your application. Methods of this class decorated with *Command* are automatically added as sub command to the application.

Basic example defining an echo command:

```
from precept import Precept, Command, Argument

class App(Precept):
    @Command(
        Argument('name'),
    )
    async def echo(self, name):
        print(f'Hello {name}')
```

Then call from the terminal like this: `app echo bob` -> Prints `Hello bob`

Note: If no command was supplied, `main` will be called instead.

1.2.1 Starting the application

To create a console application from a precept app you need to add function that will create an instance of your precept subclass and call start then add it to `setup.py`.

app.py

```
def cli():
    App().start()
```

setup.py

```
from setuptools import setup

setup(
    entry_points: {
        'console_scripts': ['cli = app:cli']
    }
)
```

You can also create a global instance of the app and assign the entrypoint to it's start method

1.3 Configs

Precept comes with a built in config system, create a subclass of *Config* with members as config. You can nest classes definition to create sub sections.

1.3.1 Example

```
from precept import Config, Nestable, ConfigProperty

class MyConfig(Config):
    my_config = ConfigProperty(comment='comment', config_type=str)

    class SubConfig(Nestable):
        nested = ConfigProperty(default='Default')

    sub_config: SubConfig  # A class member will be auto created.
```

Then you use it in the precept class like so:

```
from precept import Precept

class MyApp(Precept):
    config = MyConfig()
```

1.3.2 Config file

To use the config with files, add a `config_file` argument to precept init:

```
from precept import Precept

class MyApp(Precept):
    def __init__(self):
        super().__init__(
            config_file='config.yml',
        )
```

Precept will automatically add a `--config-file` global argument for the user to override.

It will also add a `dump-config` command to dump the default config for first use.

Note: The `config_file` argument can also be a list, in which case the first file found will be used.

1.3.3 Config format

Precept can read and write three config format, default being yaml:

ConfigFormat

- yaml
- ini
- json, doesn't support comments.

See also:

Concepts

CHAPTER

TWO

CONCEPTS

These concepts can be used to extend and interact with precept applications.

2.1 Events

Generic asyncio event consumer system.

The Precept instance and services both comes with an event dispatcher (`events` member). You can subscribe to events and handle them when they happen.

Subscribe to events

Subscribe to events that are dispatched.

```
app.events.subscribe('cli_started', lambda e: print('Started'))
```

Dispatch events

Send events with optional payload as keyword arguments.

```
app.events.dispatch('my_events', something='foo')
```

2.1.1 cli events

These events are available when starting the application with *start*.

Table 1: Cli events

Event	Description
`before_cli_start`	Called before everything else.
`cli_parsed`	Called after parsing the arguments and before the command start, payload with the arguments.
`cli_started`	The command has started, this is called at the same time.
`cli_stopped`	The command has stopped and the application will quit after.

See also:

- *Event*
- *EventDispatcher*
- *PreceptEvent*

2.2 Services

Service's runs alongside the precept application, they are started and stopped at the same time when calling *start()*.

2.2.1 Service events

Services comes with three auto events starting with the service name. If a service is called *dummy*, it will get:

- `dummy_setup`, called with the running application before starting.
- `dummy_start`, started before the application.
- `dummy_stop`, stopped after the application.

Note: If the application is not started with `start`, you need to call the services methods:

- `setup_services`
- `start_services`
- `stop_services`

See also:

- *Service*

2.3 Plugins

Plugins automatically connect to precept applications during initialisation.

They have one method, *setup()*, which takes the application as argument you can use to set variables.

To add a plugin, you need to set it in *setup.py*, the entrypoint key needs to be the snake cased version of `prog_name` variable of the precept application.

```
from setuptools import setup

setup(
    entry_points={
        'precept_app.plugins': ['my_plugin = plug:plugin']
    }
)
```

See also:

- *Plugin*

CHAPTER

THREE

APPLICATION LIST

These applications are created with precept:

- aiocast, stream videos to chromecast devices.
- top-drawer, generate valid pypi/npm package name from synonyms.

To add your application to this list, modify `docs/applications.rst` and open a PR.

CHAPTER

FOUR

PRECEPT

4.1 precept package

class precept.**Argument**(**flags: str*, *type: Optional[type] = None*, *help: Optional[str] = None*, *choices: Optional[Iterable] = None*, *default: Optional[Any] = None*, *nargs: Optional[Union[str, int]] = None*, *action: Optional[str] = None*, *required: Optional[bool] = None*, *metavar: Optional[str] = None*, *dest: Optional[str] = None*)

Bases: *precept._immutable.ImmutableDict*

Argument of a Command, can either be optional or not depending on the flags

See also:

https://docs.python.org/3/library/argparse.html#the-add-argument-method

__init__(**flags: str*, *type: Optional[type] = None*, *help: Optional[str] = None*, *choices: Optional[Iterable] = None*, *default: Optional[Any] = None*, *nargs: Optional[Union[str, int]] = None*, *action: Optional[str] = None*, *required: Optional[bool] = None*, *metavar: Optional[str] = None*, *dest: Optional[str] = None*)

Parameters

- **flags** – How to call the argument, prefixing with `-` makes the argument a keyword. Example: `'-d', '--date'` make the variable available as `date`, but can be supplied as `-d` from the cli.
- **type** – The type of the variable to cast to, default to str.
- **help** – Description to go along with
- **choices** – The available choices to choose from.
- **default** – The value to take if not supplied.
- **nargs** – Number of times the argument can be supplied.
- **action** – What to do with the argument.
- **required** – Makes a keyword argument required.
- **metavar** – Name in help.
- **dest** – The name of the variable to add the value to once parsed.

action

choices

default

dest

property flag_key

flags

help

metavar

nargs

register(*parser*)
Register a virtual subclass of an ABC.

Returns the subclass, to allow usage as a class decorator.

required

type

class precept.**AsyncExecutor**(*loop=None*, *executor=None*, *max_workers=None*)
Bases: object

Execute functions in a Pool Executor

__init__(*loop=None*, *executor=None*, *max_workers=None*)

Parameters

- **loop** – asyncio event loop.
- **executor** – Set to use an already existing PoolExecutor, default to a new ThreadPoolExecutor if not supplied.
- **max_workers** – Max workers of the created ThreadPoolExecutor.

async execute(*func*, **args*, ***kwargs*)
Execute a sync function asynchronously in the executor.

Parameters

- **func** – Synchronous function.
- **args** – Argument to give to the function.
- **kwargs** – Keyword arguments to give to the function

Returns

async execute_with_lock(*func*, **args*, ***kwargs*)
Acquire lock before executing the function.

Parameters

- **func** – Synchronous function.
- **args** –
- **kwargs** –

Returns

wraps(*func*)
Wraps a synchronous function to execute in the pool when called, making it async.

Parameters **func** – The function to wraps

Returns Async wrapped function.

class precept.**AutoNameEnum**(*value*)

Bases: enum.Enum

An enumeration.

class precept.**Command**(**arguments:* precept._cli.Argument, *name: Optional[str] = None*, *description: Optional[str] = None*, *help: Optional[str] = None*, *auto: bool = False*, *services: Optional[List[*precept._services.Service*]] = None*)

Bases: object

Command decorator, methods of *CliApp* subclasses decorated with this gets a sub-command in the parser.

Wrapped methods will gets the arguments by the Argument flag.

__init__(**arguments:* precept._cli.Argument, *name: Optional[str] = None*, *description: Optional[str] = None*, *help: Optional[str] = None*, *auto: bool = False*, *services: Optional[List[*precept._services.Service*]] = None*)

arguments: Iterable[*precept._cli.Argument*]

property command_name

description: str

register(*subparsers*)

class precept.**Config**(*config_format:* precept._configs.ConfigFormat = *ConfigFormat.TOML*, *root_name='config'*)

Bases: *precept._configs.Nestable*

Root config class, assign ConfigProperties as class members.

__init__(*config_format:* precept._configs.ConfigFormat = *ConfigFormat.TOML*, *root_name='config'*)

property config_format: *precept._configs.ConfigFormat*

read_dict(*data: dict*)

read_file(*path: str*)

save(*path: str*)

class precept.**ConfigFormat**(*value*)

Bases: *precept._tools.AutoNameEnum*

Available formats to use with configs.

- TOML provided by tomlkit, supports comments and more complex types.
- YML provided by ruamel.yaml, supports comments and more complex types.
- JSON stdlib, no support for comments and types.
- INI stdlib, support for comments.

INI = 'ini'

JSON = 'json'

TOML = 'toml'

YML = 'yml'

serializer(*config*)

class precept.**ConfigProperty**(*default=None*, *comment=None*, *config_type=None*, *environ_name=None*, *auto_environ=False*, *name=None*, *auto_global=False*, *global_name=None*)

Bases: object

__init__(*default=None*, *comment=None*, *config_type=None*, *environ_name=None*, *auto_environ=False*, *name=None*, *auto_global=False*, *global_name=None*)

class precept.**ImmutableDict**(***kwargs*)

Bases: collections.abc.Mapping

__init__(***kwargs*)

class precept.**ImmutableMeta**(*name*, *bases*, *attributes*)

Bases: abc.ABCMeta

class precept.**ImmutableProp**

Bases: object

class precept.**Nestable**(*parent=None*, *parent_len=0*)

Bases: collections.abc.Mapping

__init__(*parent=None*, *parent_len=0*)

get_prop_paths(*parent=''*)

get_root(*current=None*)

class precept.**Plugin**

Bases: object

Plugin's are automatically added to a precept application upon installation.

Set the entry point in setup to register the plugin:

entry_point = {'{app_name}.plugins': ['plugin = my_plugin:plugin']}

name: str = ''

async setup(*application*)

Setup the plugin

Parameters **application** (precept.Precept) – The running precept application.

Returns

class precept.**Precept**(*config_file: Optional[Union[str, List[str]]] = None*, *loop=None*, *executor=None*, *executor_max_workers=None*, *add_dump_config_command=False*, *help_formatter=<class 'precept._cli.CombinedFormatter'>*, *logger_level=20*, *logger_fmt=None*, *logger_datefmt=None*, *logger_stream=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*, *logger_colors=None*, *logger_style='%'*, *services: Optional[List[precept._services.Service]] = None*, *print_version: bool = True*)

Bases: object

Auto cli generator, methods decorated with Command will have a corresponding sub-command in the cli application.

Commands will get the arguments named as the last element of the command flags.

Override *main* method for root handler, it gets all the *global_arguments*

__init__(*config_file: Optional[Union[str, List[str]]] = None*, *loop=None*, *executor=None*, *executor_max_workers=None*, *add_dump_config_command=False*, *help_formatter=<class 'precept._cli.CombinedFormatter'>*, *logger_level=20*, *logger_fmt=None*, *logger_datefmt=None*, *logger_stream=<_io.TextIOWrapper name='<stderr>' mode='w' encoding='UTF-8'>*, *logger_colors=None*, *logger_style='%'*, *services: Optional[List[precept._services.Service]] = None*, *print_version: bool = True*)

Parameters

- **config_file** – Path to the default config file to use. Can be specified with `--config-file`
- **loop** – Asyncio loop to use.
- **executor** – concurrent executor to use.
- **add_dump_config_command** – Add a `dump-config` command.
- **help_formatter** – The cli formatter to use.
- **logger_level** – Set logger level when setting up logging.
- **logger_fmt** – The format of the logger.
- **logger_datefmt** – Date format of the logger.
- **logger_stream** – The stream to print the logs.
- **logger_colors** – Dictionary with key logger level name and values of bg/fg/style dict.
- **logger_style** – The symbol to use for formatting.
- **services** – List of global services to start with the program.
- **print_version** – Print the version & name of the app before start.

config: *precept._configs.Config* = None

config_class = None

property config_path

default_configs: dict = {}

global_arguments = []

async main(***kwargs*)

Handler when no command has been entered. Gets the globals arguments.

Parameters **kwargs** – Global arguments.

Returns

prog_name = ''

async setup_plugins()

Load and setup the registered plugins.

To register a plugin, subclass `Plugin` and instantiate then add to `setup.py` entry_points:

'{app_name}.plugins': ['my_plugin = plugin_module:plugin']

Returns

async setup_services(*command: Optional[precept._cli.Command] = None*)

Setup the services for the command or the main application.

Parameters **command** – The command that was run.

Returns

start(*args=None*)
Start the application loop.

Returns

async start_services(*command: Optional[precept._cli.Command] = None*)
Start the services, automatically called by start.

If the application if run with another method you can call this to start the global services without the command argument.

Parameters **command** – The command that was run.

Returns

async stop_services(*command: Optional[precept._cli.Command] = None*)
Stop the services, automatically called by start.

Call this if your application is not run with start and you have running services.

Parameters **command** – The command that was run.

Returns

version = '0.0.1'

class precept.**Service**(*events: Optional[precept.events._dispatcher.EventDispatcher] = None*)
Bases: `object`

Service's runs alongside the main application.

Communicate via events.

Events

- `{name}_setup` when added to services.
- `{name}_start` after calling start.
- `{name}_stop` after calling stop.

__init__(*events: Optional[precept.events._dispatcher.EventDispatcher] = None*)

name: str = 'service'

async setup(*application*)
Called when added to services.

Use this to set events and other post initialization that may need the application instance.

Parameters **application** (`precept.Precept`) – Precept application

Returns

async start()
Start the service.

Returns

async stop()
Stop the service.

Returns

precept.**config_factory**(*data*, *root=None*, *key=None*)

precept.**is_windows**()

4.1.1 Errors

exception precept.errors.**ConfigError**
Bases: *precept.errors.PreceptError*

Error in the config system.

exception precept.errors.**ImmutableError**
Bases: *precept.errors.PreceptError*

Immutable properties cannot change

exception precept.errors.**PreceptError**
Bases: Exception

Base exception thrown by precept

4.1.2 Subpackages

4.1.2.1 precept.console

class precept.console.**KeyHandler**(*handlers*, *loop=None*, *default_handler=None*)
Bases: object

__init__(*handlers*, *loop=None*, *default_handler=None*)

async handle()

print_keys(*file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

read()

stop()

class precept.console.**Keys**
Bases: object

BACKSPACE = <Key 'backspace'>

CTRL_A = <Key 'ctrl-a'>

CTRL_ALT_A = <Key 'ctrl-alt-a'>

CTRL_ALT_DEL = <Key 'ctrl-alt-del'>

CTRL_B = <Key 'ctrl-b'>

CTRL_C = <Key 'ctrl-c'>

CTRL_D = <Key 'ctrl-d'>

CTRL_E = <Key 'ctrl-e'>

CTRL_F = <Key 'ctrl-f'>

CTRL_Z = <Key 'ctrl-z'>

DELETE = <Key 'delete'>

DOWN = <Key 'down'>

END = <Key 'end'>

ENTER = <Key 'enter'>

ESCAPE = <Key 'escape'>

F1 = <Key 'F1'>

F10 = <Key 'F10'>

F11 = <Key 'F11'>

F12 = <Key 'F12'>

F2 = <Key 'F2'>

F3 = <Key 'F3'>

F4 = <Key 'F4'>

F5 = <Key 'F5'>

F6 = <Key 'F6'>

F7 = <Key 'F7'>

F8 = <Key 'F8'>

F9 = <Key 'F9'>

HOME = <Key 'home'>

INSERT = <Key 'insert'>

LEFT = <Key 'left'>

RIGHT = <Key 'right'>

SPACE = <Key 'space'>

SPECIAL_KEYS = (<Key 'space'>, <Key 'backspace'>, <Key 'enter'>, <Key 'escape'>, <Key 'insert'>, <Key 'end'>, <Key 'home'>, <Key 'delete'>, <Key 'down'>, <Key 'up'>, <Key 'left'>, <Key 'right'>, <Key 'F1'>, <Key 'F2'>, <Key 'F3'>, <Key 'F4'>, <Key 'F5'>, <Key 'F6'>, <Key 'F7'>, <Key 'F8'>, <Key 'F9'>, <Key 'F10'>, <Key 'F11'>, <Key 'F12'>, <Key 'ctrl-c'>, <Key 'ctrl-a'>, <Key 'ctrl-alt-a'>, <Key 'ctrl-alt-del'>, <Key 'ctrl-b'>, <Key 'ctrl-d'>, <Key 'ctrl-e'>, <Key 'ctrl-f'>, <Key 'ctrl-z'>)

UP = <Key 'up'>

classmethod get_key(*value*, *default=None*)

```
keys = {'\x01': <Key 'ctrl-a'>, '\x02': <Key 'ctrl-b'>, '\x03': <Key 'ctrl-c'>,
'\x04': <Key 'ctrl-d'>, '\x05': <Key 'ctrl-e'>, '\x06': <Key 'ctrl-f'>, '\r':
<Key 'enter'>, '\x1a': <Key 'ctrl-z'>, '\x1b': <Key 'escape'>, '\x1b\x01': <Key
'ctrl-alt-a'>, '\x1bO15~': <Key 'F5'>, '\x1bO17~': <Key 'F6'>, '\x1bO18~': <Key
'F7'>, '\x1bO19~': <Key 'F8'>, '\x1bO20~': <Key 'F9'>, '\x1bO21~': <Key 'F10'>,
'\x1bO23~': <Key 'F11'>, '\x1bO24~': <Key 'F12'>, '\x1bOP': <Key 'F1'>, '\x1bOQ':
<Key 'F2'>, '\x1bOR': <Key 'F3'>, '\x1bOS': <Key 'F4'>, '\x1b[2~': <Key 'insert'>,
'\x1b[3^': <Key 'ctrl-alt-del'>, '\x1b[3~': <Key 'delete'>, '\x1b[A': <Key 'up'>,
'\x1b[B': <Key 'down'>, '\x1b[C': <Key 'right'>, '\x1b[D': <Key 'left'>, '\x1b[F':
<Key 'end'>, '\x1b[H': <Key 'home'>, ' ': <Key 'space'>, '0': <Key '0'>, '1':
<Key '1'>, '2': <Key '2'>, '3': <Key '3'>, '4': <Key '4'>, '5': <Key '5'>, '6':
<Key '6'>, '7': <Key '7'>, '8': <Key '8'>, '9': <Key '9'>, 'A': <Key 'A'>, 'B':
<Key 'B'>, 'C': <Key 'C'>, 'D': <Key 'D'>, 'E': <Key 'E'>, 'F': <Key 'F'>, 'G': <Key
'G'>, 'H': <Key 'H'>, 'I': <Key 'I'>, 'J': <Key 'J'>, 'K': <Key 'K'>, 'L': <Key
'L'>, 'M': <Key 'M'>, 'N': <Key 'N'>, 'O': <Key 'O'>, 'P': <Key 'P'>, 'Q': <Key
'Q'>, 'R': <Key 'R'>, 'S': <Key 'S'>, 'T': <Key 'T'>, 'U': <Key 'U'>, 'V': <Key
'V'>, 'W': <Key 'W'>, 'X': <Key 'X'>, 'Y': <Key 'Y'>, 'Z': <Key 'Z'>, 'a': <Key
'a'>, 'b': <Key 'b'>, 'c': <Key 'c'>, 'd': <Key 'd'>, 'e': <Key 'e'>, 'f': <Key
'f'>, 'g': <Key 'g'>, 'h': <Key 'h'>, 'i': <Key 'i'>, 'j': <Key 'j'>, 'k': <Key
'k'>, 'l': <Key 'l'>, 'm': <Key 'm'>, 'n': <Key 'n'>, 'o': <Key 'o'>, 'p': <Key
'p'>, 'q': <Key 'q'>, 'r': <Key 'r'>, 's': <Key 's'>, 't': <Key 't'>, 'u': <Key
'u'>, 'v': <Key 'v'>, 'w': <Key 'w'>, 'x': <Key 'x'>, 'y': <Key 'y'>, 'z': <Key
'z'>, '\x7f': <Key 'backspace'>}
```

precept.console.**colorize**(*text*, *bg=None*, *fg=None*, *style=None*)

precept.console.**format_table**(*data*, *formatting=None*)

precept.console.**goto_xy**(*stream*, *x*, *y*)

precept.console.**print_table**(*data*, *formatting=None*, *file=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*)

async precept.console.**progress_bar**(*value_func*, *max_value*, *value_formatter=None*, *include_value=True*, *dents=50*, *sleep_time=0.005*, *file=None*, *full_symbol='#'*, *empty_symbol='-'*, *start_symbol='['*, *end_symbol=']'*)

async precept.console.**spinner**(*condition*, *sleep_time=0.25*, *message=''*, *fg='\x1b[37m'*, *bg=None*, *symbols=('|', '/', '-', '\\', '|', '/', '-', '\\')*)

4.1.2.2 precept.events

class precept.events.**Event**(*name: str*, *payload: dict*)

Bases: object

Event with payload and stop property.

__init__(*name: str*, *payload: dict*)

class precept.events.**EventDispatcher**

Bases: object

Dispatch events to subscribers functions.

__init__()

async dispatch(*event: str*, ***payload*)

Dispatch an event with optional payload data.

Parameters

- **event** – Name of the event.
- **payload** – Data of the event.

Returns

subscribe(*event: str*, *func*)
Subscribe func to execute every time event is dispatched.

Parameters

- **event** – The event to subscribe to.
- **func** – The func to call when event is dispathed.

Returns

class precept.events.**PreceptEvent**(*value*)
Bases: *precept._tools.AutoNameEnum*

Precept cli events.

BEFORE_CLI_START = 'before_cli_start'

CLI_PARSED = 'cli_parsed'

CLI_STARTED = 'cli_started'

CLI_STOPPED = 'cli_stopped'

PYTHON MODULE INDEX

p

precept, 9
precept.console, 15
precept.errors, 15
precept.events, 17

INDEX

Symbols

__init__() (*precept.Argument method*), 9
__init__() (*precept.AsyncExecutor method*), 10
__init__() (*precept.Command method*), 11
__init__() (*precept.Config method*), 11
__init__() (*precept.ConfigProperty method*), 12
__init__() (*precept.ImmutableDict method*), 12
__init__() (*precept.Nestable method*), 12
__init__() (*precept.Precept method*), 12
__init__() (*precept.Service method*), 14
__init__() (*precept.console.KeyHandler method*), 15
__init__() (*precept.events.Event method*), 17
__init__() (*precept.events.EventDispatcher method*), 17

A

action (*precept.Argument attribute*), 9
Argument (*class in precept*), 9
arguments (*precept.Command attribute*), 11
AsyncExecutor (*class in precept*), 10
AutoNameEnum (*class in precept*), 11

B

BACKSPACE (*precept.console.Keys attribute*), 15
BEFORE_CLI_START (*precept.events.PreceptEvent attribute*), 18

C

choices (*precept.Argument attribute*), 9
CLI_PARSED (*precept.events.PreceptEvent attribute*), 18
CLI_STARTED (*precept.events.PreceptEvent attribute*), 18
CLI_STOPPED (*precept.events.PreceptEvent attribute*), 18
colorize() (*in module precept.console*), 17
Command (*class in precept*), 11
command_name (*precept.Command property*), 11
Config (*class in precept*), 11
config (*precept.Precept attribute*), 13
config_class (*precept.Precept attribute*), 13
config_factory() (*in module precept*), 14
config_format (*precept.Config property*), 11
config_path (*precept.Precept property*), 13
ConfigError, 15
ConfigFormat (*class in precept*), 11
ConfigProperty (*class in precept*), 11
CTRL_A (*precept.console.Keys attribute*), 15
CTRL_ALT_A (*precept.console.Keys attribute*), 15
CTRL_ALT_DEL (*precept.console.Keys attribute*), 15
CTRL_B (*precept.console.Keys attribute*), 15
CTRL_C (*precept.console.Keys attribute*), 15
CTRL_D (*precept.console.Keys attribute*), 15
CTRL_E (*precept.console.Keys attribute*), 15
CTRL_F (*precept.console.Keys attribute*), 15
CTRL_Z (*precept.console.Keys attribute*), 15

D

default (*precept.Argument attribute*), 9
default_configs (*precept.Precept attribute*), 13
DELETE (*precept.console.Keys attribute*), 15
description (*precept.Command attribute*), 11
dest (*precept.Argument attribute*), 9
dispatch() (*precept.events.EventDispatcher method*), 17
DOWN (*precept.console.Keys attribute*), 15

E

END (*precept.console.Keys attribute*), 15
ENTER (*precept.console.Keys attribute*), 15
ESCAPE (*precept.console.Keys attribute*), 16
Event (*class in precept.events*), 17
EventDispatcher (*class in precept.events*), 17
execute() (*precept.AsyncExecutor method*), 10
execute_with_lock() (*precept.AsyncExecutor method*), 10

F

F1 (*precept.console.Keys attribute*), 16
F10 (*precept.console.Keys attribute*), 16
F11 (*precept.console.Keys attribute*), 16
F12 (*precept.console.Keys attribute*), 16
F2 (*precept.console.Keys attribute*), 16
F3 (*precept.console.Keys attribute*), 16
F4 (*precept.console.Keys attribute*), 16
F5 (*precept.console.Keys attribute*), 16
F6 (*precept.console.Keys attribute*), 16

F7 (*precept.console.Keys attribute*), 16
F8 (*precept.console.Keys attribute*), 16
F9 (*precept.console.Keys attribute*), 16
flag_key (*precept.Argument property*), 10
flags (*precept.Argument attribute*), 10
format_table() (*in module precept.console*), 17

G

get_key() (*precept.console.Keys class method*), 16
get_prop_paths() (*precept.Nestable method*), 12
get_root() (*precept.Nestable method*), 12
global_arguments (*precept.Precept attribute*), 13
goto_xy() (*in module precept.console*), 17

H

handle() (*precept.console.KeyHandler method*), 15
help (*precept.Argument attribute*), 10
HOME (*precept.console.Keys attribute*), 16

I

ImmutableDict (*class in precept*), 12
ImmutableError, 15
ImmutableMeta (*class in precept*), 12
ImmutableProp (*class in precept*), 12
INI (*precept.ConfigFormat attribute*), 11
INSERT (*precept.console.Keys attribute*), 16
is_windows() (*in module precept*), 14

J

JSON (*precept.ConfigFormat attribute*), 11

K

KeyHandler (*class in precept.console*), 15
Keys (*class in precept.console*), 15
keys (*precept.console.Keys attribute*), 16

L

LEFT (*precept.console.Keys attribute*), 16

M

main() (*precept.Precept method*), 13
metavar (*precept.Argument attribute*), 10
module
    precept, 9
    precept.console, 15
    precept.errors, 15
    precept.events, 17

N

name (*precept.Plugin attribute*), 12
name (*precept.Service attribute*), 14
nargs (*precept.Argument attribute*), 10
Nestable (*class in precept*), 12

P

Plugin (*class in precept*), 12
precept
    module, 9
Precept (*class in precept*), 12
precept.console
    module, 15
precept.errors
    module, 15
precept.events
    module, 17
PreceptError, 15
PreceptEvent (*class in precept.events*), 18
print_keys() (*precept.console.KeyHandler method*), 15
print_table() (*in module precept.console*), 17
prog_name (*precept.Precept attribute*), 13
progress_bar() (*in module precept.console*), 17

R

read() (*precept.console.KeyHandler method*), 15
read_dict() (*precept.Config method*), 11
read_file() (*precept.Config method*), 11
register() (*precept.Argument method*), 10
register() (*precept.Command method*), 11
required (*precept.Argument attribute*), 10
RIGHT (*precept.console.Keys attribute*), 16

S

save() (*precept.Config method*), 11
serializer() (*precept.ConfigFormat method*), 11
Service (*class in precept*), 14
setup() (*precept.Plugin method*), 12
setup() (*precept.Service method*), 14
setup_plugins() (*precept.Precept method*), 13
setup_services() (*precept.Precept method*), 13
SPACE (*precept.console.Keys attribute*), 16
SPECIAL_KEYS (*precept.console.Keys attribute*), 16
spinner() (*in module precept.console*), 17
start() (*precept.Precept method*), 14
start() (*precept.Service method*), 14
start_services() (*precept.Precept method*), 14
stop() (*precept.console.KeyHandler method*), 15
stop() (*precept.Service method*), 14
stop_services() (*precept.Precept method*), 14
subscribe() (*precept.events.EventDispatcher method*), 18

T

TOML (*precept.ConfigFormat attribute*), 11
type (*precept.Argument attribute*), 10

U

UP (*precept.console.Keys attribute*), 16

V

version (*precept.Precept attribute*), 14

W

wraps() (*precept.AsyncExecutor method*), 10

Y

YML (*precept.ConfigFormat attribute*), 11