Advanced Configuration Manager Documentation

Release 0.9

Dan Strohl

June 03, 2016

Contents

This is a configuration management plugin that helps with complex applications managing their basic settings. Often, these settings are stored as global vars in setup.py, settings.py, __init__.py files, but this does not scale well, nor is it easilly managable from within an app from a conifugration screen. This plugin helps with all of this.

Contents:

CHAPTER 1

Introduction

The advanced configuration manager came from working on a project and realizing how important it was to deal with configuration and settings, how many different places those settings were stored, how how complex it was to handle them. I started out using a collection of tools (mostly execllent) to do this, including CLINT for CLI, ConfigParser for configuration files, setting and setup files for global vars, and various custom scripts for other things. I found that I was often pulling the same kinds of things together, but there did not seem to be a package out there that handled the entire thing.

1.1 Features

The Advanced Configuration Manager will provide the following features:

- defining configuration options within your modules
- ease of access to options within your system
- support for plugins to your system
- storage of configuration options in multiple places
- handling configuration options read from multiple places
- default options
- user changable options
- locked options (non-changeable)
- entry validation
- interpolation of variables
- clean API for adding features
- upgrade / downgrade scripts for fixing configs

The API allows for easilly adding the following items:

- new configuration datatypes
- new validations
- new storage methods and systems

1.2 Requirements

At this point, this only works with python 3.4. It should be pretty easy to backport it to other 3.x versions, and I dont think it woudl be a major job to port to 2.x if there was enough call for it.

Todo

- logging and auditing
- lightweight configuration reader, heavy manager.
- support for authentication systems

CHAPTER 2

Concepts

The Advanced Configuration Manager uses the following concepts

2.1 Configuration Manager

This is the main object for the configuration manager. All interaction would normally be with this object.

2.2 Sections

A configuration is normally made up of sections, each section contains the options that are conifgured. You can also configure the system to not need sections, in which case everything will be in a single flat dictionary like structure.

2.3 Options

The options are the end nodes of the system. These are where you store your options, defaults, etc. If you setup a default they will return that default if no other value is set.

2.3.1 Interpolation

Interpolation allows one configuration variable to reference another. so, for example, you could have options "first_name" and "last_name", and have a third option with a value of "%(first_name) %(last_name)", which, when polled would return the two names combined.

2.4 Storage

This is where the options are stored. There are various storage managers included, and an API for creating more. For example, in a typical option set, you may have two storage managers used. a file storage manager for saving text files and the cli manager for over-ridding some of the options with cli parameters. In a more complex environment, you might have three storage managers used. the cli storage to define where the startup file is stored, a local text file to configure the database connection parameters, and a database storage manager to save the majority of the items.

You might even have other storage managers setup for specific items such as user specific settings, group or team settings, etc...

You can even use an initial storage and configuration file to configure later storage managers (such as the above mentioned database managers).

2.4.1 Storage Names

Most of this is handled by setting storage names. Each storage manager has a name. each section has two parameters that control what storage they can use.

storage_write_to can have the following settings:

- '<name>': The name of the storage location to save to.
- None: The section will be saved to the default location.
- **'-': The section will NOT be saved in save_all operations (though it can still be manually saved to a storage** location)
- **"*": The section will be saved to ALL configured standard storage locations. (not recommended in complex** environments)

storage_read_from_only:

- **['<name>', '<name>']: A list of names, this will only allow data read from these storage locations to be used for** these sections.
- None (default): Options in storage will be always be used.

Note: CLI options, if configured, will always overwrite data from storage.

CHAPTER 3

Using the Advanced Configuration Manager

3.1 Usage

Using the Configuration Manager can be simple, the minimum that you need to do is:

- Import `ConfigManager` and instantiate it.
- Define the options that you want to use
- Reference the options

3.1.1 Setup the Manager

```
from AdvConfigMgr import ConfigManager
config = ConfigManager()
```

3.1.2 Defining Options

```
# define the section for the options
config.add('my_section')

# define the options within the section
config['my_section'].add(my_option1='default_value', my_option2=1234, my_option3=['item1', 'item2'])
# there are more ways of setting options as well

# if no advanced features are needed, often this will be handled more like this
section_config = config['my_section']

...

section_config['option1'] = value
section_config['option2'] = another_value
section_config['option3'] = still_another_value
```

3.1.3 Using Options

```
# to save typing, we can assign the section to a variable:
sec = config['my_section']
```

```
# reading the option
temp_var = sec['my_option1']

# setting the option value
sec['my_option'] = 'new value'

# clearing the option and using the default value
del ['my_option']

# you can also do something like this
tmp_var = config['my_section']['my_option1']

# or even this if it works better for you
tmp_var = config['my_section.my_option']
```

3.1.4 Saving Options to a file

```
# saving to a storage location such as a file can be as easy as
config.write(file='myfile.ini')

# and readign it again later
config.read(file='myfile.ini')

# the filename can also be set during the initial setup, in which case you can simply do:
config.write()
config.read()
```

3.1.5 Advanced Features and configuration

There are many advanced features and configuration options that are available if needed, as well as several ways of extending the system.

3.2 Advanced Usage

For more information on how to use the advanced features of this module in your system, see the following topics.

3.2.1 CLI Argument Management

The system can handle managing arguments passed via the CLI as well as ones in config files or databases. When configuring an option, you have the ability to pass a dictionary of parameters to the option keyword "cli_option". This dictionary will tell the system to allow that option to be configured by the CLI, defines the help for the cli, and tells the system how to handle the response.

- flags: (required) a flag or list of flags to be accepted. (these must be unique across the system).

> **Note:** If only a single string is passed instead of a dictionary, it is assumed to be the flag and the rest of the data will be set at defaults.

- **data_flag: If present, this will set the "store_true" or "store_false" action in argparse, this means that** if True is set, and the cli flag is set, the option will be set to True, and vice-versa. If a default is used, it must be a boolean.
- **nargs: [int] This will allow that number of arguments to be added to a list config item, so for example,** if nargs 3 is passed, the following argument would be treated as one:

```
program.py -arg 1 3 4
```

  This would return ['1','2','3'], however if nargs 2 was passed, this would return ['1','2'] and the '3' would be treated as another positional argument. If a default is used in this case, it must be a list and have this many items.

- **choices: a list of choices that are available (and will be validated against). If a default is used, it** must be present in the list.
- **default: if not present, will use the option default, if present and None, will not have a default,** if present and set to somethign, will use that as a default. .. note:: if a default is present, either passed or using the option default, it must also fit the other arguments.
- required: [True/False] this option MUST be passed on the command line.
- **help: this is a descriotion string to be used on the command line. if this is not present,** the option description will be used.

```
cli_option: ({'flags':['f', 'foo'],
              'data_flag':False,
              'nargs':2,
              'choices':['bar','barr','b'],
              'required':False,
              'help':'This defines if this is fu or bar'})
```

3.2.2 Migration of Versions

The system can help with migrating the data between versions. To handle migrations, first create a migration definition dictionary. and pass it to the migration manager during section initialization.

Migrations are handled on a section by section basis, and are performed when running a read operation from a storage manager (other than the cli manager) So be aware that in a multi-storage environment,

This dictionary could have the following keys:

Key Name	Data Type	Description
section_name	str	The name of the section to be used. can be omited only if a simple config is used
stored_version_min	str	The version of the stored data to convert from, this re
stored_version_max	str	version entries, and could have all three defined.
stored_version	str	If stored_version is None, (as oposed to omitted), this
live_version_min	str	from stored configs that do not have a version defined
live_version_max	str	The version of the stored data to convert from, this re
live_version	str	
actions	list of actions	A list of actions with the option names and the actions to take for each.
keep_only	bool	If True, only options in the list will be kept, for record based storage all others will be deleted (Defaults to False
	ActionClass	a subclass of `BaseMigrationActions` that is used to do the changes

Actions action_class

This is a list of the actions that you want taken, each action record (dictionary or tuple) defines how you want to handle an option (or set of options matching the filter).

The action records are either a dictionary of keys, or a tuple, these are passed to the action method as either a set of args (if a tuple) or as a kward set (if a dictionary)

the action methods are in the MigrationActionClass, which you can also subclass and pass to in the migration definition dictionary if you want to handle migrations differently.

An example of this is would be, if there is a migration action ('remove','my_option') the MigrationManager class will call MigrationActionClass.remove('my_option'). This class method is responsible for carrying out the migration actions.

This could also have been passed as an action {'action':'remove','option_name':'my_option'}

Remove

These options will be removed and deleted from storage. [1]

Dict:

```
{'action':'remove','option_name':<option_name>}
```

Tuple:

```
('remove','option_name')
```

Rename

This will rename the options. you have the option to pass an interpolation string as well with this that can modify the value of the string type options.

[1] Only record based storage managers will delete these from the storage medium. others will rely on the object not being present in the config since the entire config is re-written to the storage overwriting the old one.

Dict:

```
{'action':'rename',
'option_name':<old option name>,
'new_option_name':<new option name>,
['interpolation_str':<optional interpolation string>]}
```

Tuple:

```
('rename','old_option_name','new_option_name'[, 'interpolation_str'])
```

Copy

This will copy an option and add it from another option. This also allows copying from another section if a name is passed using dot notation as well as an optional interpolation string.

Dict:

```
{'action':'copy',
'option_name':<existing option name>,
'new_option_name',<new option name>,
['interpolation_str':<interpolation_string>]}

{'action':'copy',
'option_name':<existing section name.existing option name>,
'new_option_name',<new option name>,
['interpolation_str':<interpolation_string>]}
```

Tuple:

```
('copy','old_option_name','new_option_name' [,'interpolation_str'])
('copy','old_section.old_option_name,'new_option_name' [, 'interpolation_str'])
```

Pass

These will just be passed through, this is generally not needed unless the 'keep_only' flag is used, then these are required for all options to be kept.

Dict:

```
{'action':'pass',
'option_name':<option_name>)
```

Tuple:

('pass','option_name')

Interpolate

This runs the option through a simple interpolator, allowing for some simple conversions.

Dict:

```
{'action':'interpolate',
'option_name':<option_name>
['interpolation_str':<interpolation string>]}
```

Tuple:

```
('interpolate','option_name' [,'interpolation_str'])
```

Other

For more complex conversion needs, you can subclass the `BaseMigrationActions` class and create your own migrations. Anything that is not in the above list will be passed to a method of the MigrationActions class method matching the name of the action. every action definition must have a 'action' key (or the first item in the tuple, and an 'option_name' key (the second item in the tuple) that defines what options are sent. along with the defined keys a kwarg of 'value' will also be passed that is the current value of the option.

Some actions support glob type wildcards ('*', '?', '!', '[]'), (by default the 'remove', 'interpolate', and 'pass' ones) Generally these would be ones that do not require changing the option name).

For interpolations, use '%(__current_value__)' for the current value, %(option_name), %(section_name.option_name) to pull in othe values.

By default, options without actions will simply be passed through unless the "keep_only" flag is set.

3.2.3 Validation of data

If the data is going to be manipulated by users, the data can also be validated for accuracy. This is done by

3.2.4 Managing Data Storage Locations

Ths system can support multiple storage locations for the data, including handling data stored on local disk INI files, databases, and even passing the data in dictionary or list format.

3.2.5 Interpolation Options

The interpolation system can handles replacement on the fly. This is done within the options themselves, as well as during migrations.

At its most basic, interpolation allows you to have a string that contains a key string that is replaced with the value of another configuration option. For example:

:: # setup the configuration manager

config = ConfigManager() config.add('section1') section1 = config['section1']

add some options for the data directory and data file

section1['data_dir'] = 'my_path/data' section1['data_filename'] = 'my_data_file.db'

add an option that will return the full file and path for the database:

section1['database'] = '%(data_dir)/%(data_filename)'

when you call the interpolated option section1['database']

'mypath/data/my_data_file.db'

..note:: there are better approaches to handling paths in the system that woudl work better with cross platform systems, see **:py:module:'pathlib'**

Interpolation can also handle cross section variables, so you can use %(section_name.option_name) if needed, as well as recursive variables where you are including a interpolated option in another interpolated option (up to a max of 10 levels).

if you need to use "%" in your option values, you can use escape it with "%%"

..note:: you can disable interpolation by passing None to the "interpolator" keyward arg for the `ConfigManager`

CHAPTER 4

Configuration and API

4.1 Core Classes

These are the core classes that you normally interact with through the configuration manager. most of the other classes, such as the Storage Manager Classes, Migration Assist Classes, etc... are called and controlled by these.

4.1.1 ConfigManager Class

class AdvConfigMgr.**ConfigManager**(*version=None*, *migrations=None*, *storage_config=None*, *storage_managers=None*, *default_storage_managers=None*)

Parameters

- **allow_no_value** – allow empty values.
- **empty_lines_in_values** –
- **allow_add_from_storage** – allow adding sections and options directly from the storage
- **allow_create_on_set** – allows sections to be created when the set command is used. the set command can only be used to set the values of options, so this REQUIRES using dot-notation.
- **no_sections** – this will disable all sections and all options will be accessible from the base manager object. (this creates a section named "default_section".) .. warning:: converting from simple configurations to sections may require manual data minipulation!
- **section_defaults** – a dictionary of settings used as defaults for all sections created
- **interpolation** – can be defined if interpolation is requested or required.
- **section_option_sep** (*str*) – defines a seperator character to use for getting options using the dot_notation style of query. defaults to '.', in which case options can be queried by calling ConfigManager[section.option] in addition to ConfigManager[section][option]. If this is set to None, dot notation will be disabled.
- **cli_program** – the name of the program for the cli help screen (by default this will use the program run to launch the app)
- **cli_desc** – the text to show above the arguments in the cli help screen.
- **cli_epilog** – the text to show at the end of the arguments in the cli help screen.

- **raise_error_on_locked_edit** – if True, will raise an error if an attempt to change locked options, if False (default) the error is suppressed and the option will not be changed.
- **storage_managers** (*BaseStorageManager*) – a list of storage managers to use, if none are passed, the configuration will not be able to be saved.
- **cli_parser_name** – the name of the cli parser if not the default. set to None if the CLI parser is not to be used.
- **cli_group_by_section** – True if cli arguments shoudl be grouped by section for help screens.
- **version_class** (*Version or str or None*) – 'loose', 'strict', a subclass of the Version class or None for no versioning.
- **version** (*str*) – the version number string.
- **version_option_name** (*str*) – the option named used to store the version for each section, {section} will be replaced by the name of the section.
- **version_allow_unversioned** (*bool*) – if True, the system will import unversioned data, if false, the version of the data must be specified when importing any data.
- **version_enforce_versioning** (*bool*) – if True, the system will raise an error if no version is set at the section or base level.
- **version_disable_cross_section_copy** (*bool*) – if True, cross section copy will not work. Used when you have plugins from different authors and you want to segment them.
- **version_make_migrations** (*list*) – this is a list of migrations that can be performed, (see `migration`)
- **kwargs** – if "no_sections" is set, all section options can be passed to the ConfigManager object.
- **data_dict** (*ConfigDict*) – any dictionary that can support the number of levels needed.
- **version** – the sering version number.
- **migrations** (*list*) – a list of migration dictionaries.
- **storage_config** (*dict*) – a dictionary of storage configuration dictionaries. These would be in the format of {'storage_name':{<config_dict>},'storage_name':{<config_dict>}}. See specific storage managers for details of the config dict entries.
- **storage_managers** – a list or set of storage managers to use, if not passed, the file storage manager is used.
- **default_storage_managers** (*list*) – a list or string indicating the default storage manager(s) names to use if no names are passed during read/write operations. if None (the default) all conifgured storage managers are polled.

add(**args*, ***kwargs*)

Adds configuration options:

```
add('section1', 'section2', 'section3')
add(section_def_dict1, section_def_dict2, section_def_dict3)
add([list of section_def_dicts])
add('section_name.option_name', 'section_name.option_name')

add(section_name1='option_name1', section_name2='option_name2')
```

```
        add(section_name=section_def_dict, section_name2=section_def_dict)
        add(section_name=[list_of_option_names or dicts], section_name=(list of option names or dict
```

section_def_dict keys

Key	Default	Description
'name'	None	The name of the section
'verbose_name'	None	The verbose name of the section
'description'	None	A long description of the section
'storage'	None	The name of the storage location (if used)
'keep_if_empty'	False	Keep the section even if all options ahve been deleted
'store_default'	False	Store defaults in storage medium
'locked'	False	If True, do not allow any changes to the section
'allow_create_on_load'	True	Allow new options to be created directly from the storage medium for example, if you hand edit the ini file and add new options
'option_defaults'	None	Allows a dict to be passed with defaults for any new options in this section this will replace any system wide option defaults specified.
'options'	None	Provides a list of options to be added to the section,

Note: When no sections are used, this will redirect to `ConfigSection.add()`

add_section(*section*, *force_add_default=False*, ***kwargs*)
Create a new section in the configuration.

Raise DuplicateSectionError if a section by the specified name already exists.

read(*sections=None*, *storage_names=None*, *override_tags=False*, *data=None*, ***kwargs*)
runs the read from storage process for the selected or configured managers

Parameters

- **storage_names** – If None, will read from all starnard storage managers, if a string or list, will read from the selected ones following the configured tag settings.
- **sections** – If None, will read from all sections, if string or list, will read from the selected ones following the configured tag settings.
- **override_tags** – if True, this will override the configured storage tag settings allowing things like exporting the full config etc.
- **data** – if a single storage tag is passed, then data can be passed to that storage manager for saving. this will raise an AssignmentError if data is not None and more than one storage tag is passed.

write(*sections=None*, *storage_names=None*, *override_tags=False*, ***kwargs*)
runs the write to storage process for the selected or configured managers

Parameters

- **storage_names** – If None, will write to all starnard storage managers, if a string or list, will write to the selected ones following the configured tag settings.
- **sections** – If None, will write to all sections, if string or list, will write to the selected ones following the configured tag settings.
- **override_tags** – if True, this will override the configured storage tag settings allowing things like exporting the full config etc.

Returns if ONLY one storage_tag is passed, this will return the data from that manager if present.

4.1.2 ConfigSection Class

class AdvConfigMgr.**ConfigSection**(*manager*, *name*, *verbose_name=None*, *description=None*, *storage_write_to=None*, *storage_read_from_only=None*, *store_default=False*, *locked=False*, *allow_create_on_load=True*, *allow_create_on_set=True*, *option_defaults=None*, *cli_section_title=None*, *cli_section_desc=None*, *version=None*, *version_option_name=None*, *version_migrations=None*, *options=None*)

A single section of a config

Parameters

- **manager** (ConfigManager) – A pointer to the config manager
- **name** (*str*) – The name of the section
- **verbose_name** (*str*) – The verbose name of the section
- **description** (*str*) – A long description of the section
- **storage_write_to** (*str*) – The tag of the storage location to save to, if None, the section will be saved to the default location, if '-' it will not be saved in save_all operations, if "*", section will be saved to all configured storage locations. Sections can be saved manually to any storage if needed.
- **storage_read_from_only** (*str or list*) – options from storage with tags in this list will be read. If None (default) then options in storage will be always be used. This allows restricting options to specific storage locations. CLI options, if configured, will always overwrite data from storage.
- **store_default** (*bool*) – store defaults in storage medium
- **locked** (*bool*) – if True, do not allow any changes to the section
- **allow_create_on_load** (*bool*) – if True, options will be created if they are in the stored data, if false, they must be configured first.
- **allow_create_on_set** (*bool*) – if True, options can be created using a dictionary set proces, if False, they need to be configured first.
- **cli_section_title** (*str*) – the name of the section in the CLI (if sectioned), Defaults to verbose_name
- **cli_section_desc** (*str*) – the description for the section in the CLI (if sectioned), Defaults to description
- **version** (*str*) – the version number of the section, if None, this will take the version number of the ConfigManager object.
- **version_option_name** (*str*) – allows for overriding the ConfigManager's version_option_name setting.

add(**args*, ***kwargs*)

Adds new option definition to the system

args, kwargs: config options can also be passed in args/kwargs, in a number of formats.

Examples:

This does not set any default values:

```
add('option1', 'option2', 'option3')
```

Seperate dictionaries:

```
add(full_option_dict, full_option_dict)
```

A list of dictionaries:

```
add([full_option_dict, full_option_dict])
```

A list of sets with option_name and default value.:

```
add([('option1',default_value),('option2',default_value)]
```

If default value is a dict, this will not work, if a dict is passed, it is assumed to be a full_option_dict:

```
add(option1=default_value1, option2=default_value2, option3=default_value3)

add(option1={full_option_dict}, option2={full_option_dict))
```

These can be mixed, so the following would be valid:

```
add('option1', full_option_dict, [full_option_dict, full_option_dict], [('option2',default_v
```

full_option_dict Example (with defaults): 'name': '<name of option>', 'default_value': _UNSET, 'datatype': None, 'verbose_name': None, 'description': None, 'cli_option': None, 'validations': None, 'do_not_change': False, 'do_not_delete': False, 'required_after_load': False,

---

Note:

•If a default value is a dictionary, it must be passed within a full option dict.

•See ConfigOption for option_dict parameters.

•If a full option dict is passed as an arg (not kwarg) it must contain a 'name' key.

•Args and kwargs can be mixed if needed... for example this is also a valid approach:

```
add(option1, <full_option_dict>, option3=default_value, option4={full_option_dict}
```

•If options are repeated in the same commane, kwargs will take precdence over args, and new options will overwrite old ones.

•if there are existing options in the section with the same name, an error will be raised.

---

clear(*options*, *forgiving=False*)

Will set the option to the default value or to unset as long as long as the section is not locked.

Parameters

- **options** (*str or list*) – option name or list of option names
- **forgiving** (*bool*) – True will return False if the option is not found, False, will raise NoOptionError.

Returns True if all deletes passed, False if not. if False, a list of the failed options is stored in self.last_failure_list

Return type bool

Raises **NoOptionError** – If the option does not exist. and forgiving is False.

delete(*options*, *force=False*, *forgiving=False*)
Will delete a list of options.

Parameters

- **options** – Option name or list of option names.
- **force** (*bool*) – True will delete the object even if it has a default_value without checking for value or lock.
- **forgiving** (*bool*) – True will return False if the option is not found, False, will raise NoOptionError.

Type str or list

Returns True if all deletes passed, False if not. if False, a list of the failed options is stored in `ConfigSection.last_failure_list`

Return type bool

from_read(*option*, *value*, *raw=False*, *validate=True*)
adds data from a storage module to the system, this ignores the 'do_not_add' flag. :param str option: option name :param value: the value to add :param bool raw: True if the interpolation needs to be bypassed. :param bool validate: True if validation should happen. :return:

get(*option*, *fallback=Empty Value*, *raw=False*)
gets the value of an option

Parameters

- **option** – the name of the option
- **fallback** – a value to return if the option is empty
- **raw** – a flag to skip interpolation

Returns

items(*raw=False*)
Return a list of (name, value) tuples for each option in a section.

All interpolations are expanded in the return values, based on the defaults passed into the constructor, unless the optional argument 'raw' is true.

Parameters **raw** (*bool*) – True if data should not be interpolated.

load(*option*, *value*, **args*, ***kwargs*)
Loads value into system, can create new options if "allow_create_on_load" is set.

Parameters

- **name** – Option Name
- **value** – Option Value
- **args** – as per ConfigOption class (passed through)
- **kwargs** – as per ConfigOption class (passed through)

section_ok_after_load
validates that all options that are required "after_load" have either a set or default value :return: :rtype boolean:

set(*option*, *value*, *raw=False*, *validate=True*, *force=False*)
Sets an option value.

Parameters

- **option** – the name of the option to set
- **value** – the value to set
- **raw** – if set to True will bypass the interpolater
- **validate** – if False will bypass the validation steps
- **force** – if True will bypass the lock checks, used for loading data.

Returns the interpolated value or default value.

to_write(*option*, *raw=False*, *as_string=False*)
gets data from the system to save to a storage module :param bool raw: :param bool as_string: :return:

4.1.3 ConfigOption Class

class AdvConfigMgr.**ConfigOption**(*section*, *name*, **args*, ***kwargs*)
An individual option in the config

Parameters

- **section** (*ConfigSection*) – A pointer to the ConfigSection object that this is a part of
- **name** (*str*) – The name of the config object. This is transformed by the optionxform method in the main config manager. by default this is converted to lowercase
- **default_value** (*object*) – Default= _UNSET the default value for the item. If set to _UNSET this is considered to not have a default. (this allows None to be a valid default setting.
- **data_type** (*str*) – Default=None: This is the type of data that is stored in the option. this accepts : None, 'str', 'int', 'float', 'list', 'dict' additional data types can be defined using the DataTypeBase class. If set to None and there is a default value set, this will take the datatype of the default value, otherwise it will be set to 'str'
- **verbose_name** (*str or None*) – Default=None This is the long name for the option (that can show up in the options configuration screen or help screen) This is set to a title case version of the option name with spaces replacing '_'
- **description** (*str or None*) – Default=None This is the long description for the option, available in the help screens.
- **cli_option** (*str or None*) – Default=None This allows the option to be changed via the CLI on startup, this would be a string, tuple or dictionary of options that configure how the cli commands will be handled.
- **validations** (*object*) – Default=None: This is a set of validation classes to be run for any options saved.
- **keep_if_empty** (*bool*) – Default=True: If set to False the option will be deleted when the value is cleared AND there is no set default value.
- **do_not_change** (*bool*) – Default=False If set to True, this will not allow the user to change the option after initial loading.
- **do_not_delete** (*bool*) – Default=False If set to True, this will not allow the user to delete the option.

- **required_after_load** (*bool*) – Default = False, If set to true, the app should not start without this being set. if there is a CLI_option available, the app should prompt the user for that option, if not, the app should fail with a usefull message.
- **autoconvert** (*bool*) – will attempt to autoconvert values to the datatype, this can be disabled if needed. (some types of data may not autoconvert correctly.)

from_read(*value*, *raw=False*, *validate=True*, *from_string=False*)
adds data from a storage module to the system, this ignores the 'do_not_add' flag.

Parameters

- **value** – the value to add
- **raw** – if set to True will bypass the interpolater
- **validate** – if False will bypass the validation steps
- **from_string** – if True will convert from string

Returns the interpolated value or default value.

get(*raw=False*, *as_string=False*)
Gets the current value or default interpolated value.

Parameters **raw** – if set to True will bypass the interpolater

Returns the interpolated value or default value.

set(*value*, *raw=False*, *validate=True*, *force=False*)
Sets the current value.

Parameters

- **value** – the value to set
- **raw** – if set to True will bypass the interpolater
- **validate** – if False will bypass the validation steps
- **force** – if True will bypass the lock checks

Returns the interpolated value or default value.

to_write(*raw=False*, *as_string=False*)
gets data from the system to save to a storage module,

Parameters

- **raw** – if set to True will bypass the interpolater
- **as_string** – returns the value as a strong (passing through the datatype module to_string method)

Returns the interpolated value or default value.

Returns the interpolated value or default value.

4.2 Data Type Classes

These classes manage the data types available in the system. They handle validation and data conversion for the system. The system comes with seveal datatypes pre-defined, but you can add more if you need special handling of your data.

4.2.1 Data Type Base Class

class AdvConfigMgr.config_types.**DataTypeBase**(*validations=None*, *allow_empty=True*, *empty_type=Empty Value*)

Parameters

- **allow_empty** – set to False if validation should be raised on empty or blank fields.
- **empty_types** – set to a tuple of types that are considered empty
- **validations** – a list or tuple of validation classes to run.

from_string(*value*, *validate=True*)
Returns an object matching the datatype from a string

to_string(*value*)
Returns a string version of the value passed.

validated(*value*)
Runs all validations and returns the value if validated. :param value: value to be validated :return:

4.2.2 Pre-Configured Validation Classes

4.3 Storage Manager Classes

The system can store configuration data in multiple storage locations, the following classes are used by the system to handle reading and writing from storage locations.

There are three types of storage locations that the system recognizes;

Read-Only: These are locations that do not ahve the ability to store data, Currently this is only the CLI manager.

Block-Based: These are locations that or write everything at once and do not have the ability to modify a single option. Examples of these are INI text files and the dictionary and string managers.

Record-Based: These are the most complex managers and are normally based on databases. These have the ability to read or write options individually.

4.3.1 ConfigStorage Class

Configuration Manager

class AdvConfigMgr.**StorageManagerManager**(*config_manager*, *managers=None*, *cli_parser_name='cli'*, *cli_manager=None*, *storage_config=None*, *default_storage_managers=None*)

A class to handle storage managers

Parameters

- **config_manager** – a link to the ConfigurationManager object
- **managers** – the managers to be registered. The first manager passed will be imported as the default
- **cli_parser_name** – the name of the cli parser if not 'cli', if None, this will disable CLI parsing.

- **cli_manager** – None uses the standard CLI Parser, this allows replacement of the default cli manager

read(*sections=None*, *storage_names=None*, *override_tags=False*, *data=None*)
runs the read from storage process for the selected or configured managers

Parameters

- **storage_names** – If None, will read from all starnard storage managers, if a string or list, will read from the selected ones following the configured tag settings.
- **sections** – If None, will read from all sections, if string or list, will read from the selected ones following the configured tag settings.
- **override_tags** – if True, this will override the configured storage name settings allowing things like exporting the full config etc.
- **data** – if a single storage name is passed, then data can be passed to that storage manager for saving. this will raise an AssignmentError if data is not None and more than one storage name is passed.

write(*sections=None*, *storage_names=None*, *override_tags=False*, ***kwargs*)
runs the write to storage process for the selected or configured managers

Parameters

- **storage_names** – If None, will write to all starnard storage managers, if a string or list, will write to the selected ones following the configured tag settings.
- **sections** – If None, will write to all sections, if string or list, will write to the selected ones following the configured tag settings.
- **override_tags** – if True, this will override the configured storage name settings allowing things like exporting the full config etc.

Returns if ONLY one storage_name is passed, this will return the data from that manager if present.

Configuration Storage Plugins

Storage Plugin Base Class

class AdvConfigMgr.**BaseConfigStorageManager**
Base class for storage managers, efines an expandable storage subsystem for configs.

Also, the two methods; BaseConfigStorageManager.read() and BaseConfigStorageManager.write() need to be overwritten to read and write the data in the format needed.

if the manager is intended to be a 'standard' one, in other words, if it will be used for automatic read-all/write-all processes, it must be able to run without passing any data or arguments, all configuration must be done during initialization. if it will only be used standalone or on-demand, you can allow any information to be passed.

allow_create = True

Parameters **force** (*bool*) – True if this will set options even if they are locked

config(*config_dict*)

Parameters **config_dict** – a dictionary with storage specific configuration options., this is called after the storage manager is loaded.

force = False

Parameters **overwrite** (*bool*) – True if this will overwrite options that have existing values

force_strings = False

Parameters **standard** (*bool*) – True if this should be used for read_all or write_all ops

lock_after_read = False

Parameters **priority** (*int*) – the priority of this manager, with smallest being run earlier than larger.

overwrite = True

Parameters **lock_after_read** (*bool*) – True if this will lock the option after reading

read(*section_name=None*, *storage_name=None*, ***kwargs*)

Read from storage and save to the system

Parameters

- **section_name** (*str or list*) – A string or list of sections to read from in the config.
- **storage_name** (*str*) – A string name of the storage manager, this can be used to override the configured name.
- **kwargs** – each storage manager may define its own additional args, but must also implement the final kwargs parameter so that if it is called with other arguments, it wont cause an error.

Returns the number of sections / options added

The recommended implementation method us to read from your storage method (database, special file, etc) and store the arguments in a dictionary or dictionary of dictionaries. then pass that dict to `BaseConfigStorageManager._save_dict()`. that method will take care of writing the data, converting it if needed, making sure that it is allowed to write, handling locked sections and options, etc...

if the implementation tries to pass data directly to the file manager for importing, it will save the data in `BaseConfigStorageManager.data()` where you can read it, so you should check this before processing.

You shoudl keep track of the number of sections and options written/read and return these at the end:

```
        return self.last_section_count, self.last_option_count
```

standard = True

Parameters **allow_create** (*bool*) – True if this can create options in the system, even if they are not pre-configured.

storage_name = None

Parameters **force_strings** (*bool*) – If True, the system will convert all options to strings before writing to the manager, and from strings when reading from it.

storage_type_name = 'Base'

Parameters **storage_name** (*str*) – The internal name of the storage manager, must be unique

write(*section_name=None*, *storage_name=None*, ***kwargs*)

Write data from the system and save to your storage

Parameters

- **section_name** (*str or list*) – A string or list of sections to write to.

- **storage_name** (*str*) – A string name of the storage manager, this can be used to override the configured name.
- **kwargs** – each storage manager may define its own additional args, but must also implement the final kwargs parameter so that if it is called with other arguments, it wont cause an error.

Returns the number of sections / options written

The recommended implementation method is to call `BaseConfigStorageManager._get_dict()` which will return a dictionary of the options or dictionary of sections (which are dicts of options) to be saved. You can then iterate through these and save them in your storage system.

if you want to return data direct from the write method, you should copy it to `BaseConfigStorageManager.data()` after processing.

you shoudl keep track of the number of sections and options written/read and return these at the end:

```
        return self.last_section_count, self.last_option_count
```

CLI Plugin Class

class AdvConfigMgr.**ConfigCLIStorage**

Read configuration from the CLI

force = True

True if this will set options even if they are locked

force_strings = False

True if the storage only accepts strings

lock_after_read = True

True if this will lock the option after reading

overwrite = True

True if this will overwrite options that have existing values

read(*section_name=None*, *storage_name='cli'*, ***kwargs*)

will take a dictionary and save it to the system :param dict_in: :param storage_name: :return:

reset_cache()

Reloades the cli_parser from the config.

standard = True

True if this should be used for read_all/write_all ops

write(*section_name=None*, *storage_name='cli'*, ***kwargs*)

cli does not accept writing options – disabled

Dictionary Plugin Class

class AdvConfigMgr.**ConfigSimpleDictStorage**

Read configuration from a dictionary.

Keys are section names, values are dictionaries with keys and values that should be present in the section.

read(*section_name=None*, *storage_name='dict'*, ***kwargs*)

will take a dictionary and save it to the system :param dict_in: :param storage_name: :return:

standard = False
True if this should be used for read_all/write_all ops

write(*section_name=None*, *storage_name='dict'*, ***kwargs*)
will return a dictionary from the system :param storage_name: :return:

File Plugin Class

class AdvConfigMgr.**ConfigFileStorage**
A file manager that stores config files in a text file

this manager can handle multiple files, as well as a string or list of data, as long as the data is in the format of an ini file. it can also handle scanning a directory or list of directories for all files matching a filter pattern.

if multiple files or filenames are passed, the files read will be processed in the order they are listed, with sections being merged and options overwriting older ones.

if a directory path is passed, the files will be sorted based on the "read_path_order" option and processed in that order.

Parameters

- **delimiters** (*tuple*) – the delimiter between the key and the value
- **comment_prefixes** (*tuple*) – this is a tuple of characters that if they occur as the first non-whitespace character of a line, the line is a comment
- **inline_comment_prefixes** (*tuple*) – this is a tuple of characters that if they occur elsewhere in the line after a whitespace char, the rest of the line is a comment.
- **space_around_delimiters** (*bool*) – True if space should be added around the delimeters.
- **strict** (*bool*) – if False, duplicate sections will be merged, if True, duplicate sections will raise an error
- **read_filenames** (*str or list*) – a filename or list of file names, assumed to be in the current directory if not otherwise specified for reading. These can also be path/globs and the system will attempt to read all files matching that glob filter. for example, the following are all exampels of valid parameters:

```
'myfile.ini'
'dir/myfile.ini'
'dir/\*.ini'
['myfile.ini', 'myotherfile.ini', 'backup_files/myfile_??.ini']
```

The filename to read from can also be passed during the read operation.

- **read_path_order** – 'alpha' (default) or 'date', the order files will be processed if a path is passed.
- **filename** (*str*) – If used, the single file to read and write to. (cannot be used with read_filenames write_filename.)
- **write_filename** (*str or None*) – the filename to write files to. if None and read_filenames is passed, this will take the first name in the list. if None and read_paths is passed, AND if there is ONLY ONE file in the path that matches the filter, this will use that file. the filename to write to can also be passed during the write operation.

- **leave_open** (*bool*) – if True, the file objects will be left open while the config manager is loaded. this can speed up file access, but it also uses up file handles, buffers, memory, and has the possibility of corrupted files.
- **create_files** (*bool*) – if False, will not create any files it does not find.
- **fail_if_no_file** (*bool*) – if False, will fail and raise an error if the specified filename is not found.
- **make_backup_before_writing** (*bool*) – if True, the system will make a backup file before writing the configuration.
- **backup_filename** (*str*) – the filename of the backup file. this can have the following formatting keys: '{NUM}' for an incremental number (uses the next available number) '{DATE}' for a date string ('YYYYMMDD') '{STIME}' for a 1 second resolution time string ('HHMMSS') '{MTIME}' for a 1 minute resolution time string {'HHMM') '{NAME}' for the old config file name (without extension)
- **backup_path** (*str*) – if not None (the default) this allows the backup file to be in a different location.
- **max_backup_number** (*int*) – the max number (assuming a backup file and NUM in the filename)
- **encoding** (*str*) –

Returns

config (*config_dict*)

Parameters **config_dict** (*dict*) – a dictionary with storage specific configuration options., This is called after the storage manager is loaded.

read (*section_name=None*, *storage_name='file'*, *files=None*, *encoding=None*, ***kwargs*)
will read an ini file and save it to the system

Parameters

- **section_name** (*str or list*) –
- **storage_name** (*str*) –
- **file** (*str or FileObject*) –
- **encoding** (*str*) –

Returns

Return type int

storage_name = 'file'
the internal name of the storage manager, must be unique

write (*section_name=None*, *storage_name='file'*, *file=None*, *encoding=None*, ***kwargs*)
will write to an INI file.

MongoDB Plugin Class

Todo

This

4.4 Validation Classes

These classes are used by the Data Type Classes to provide validation. by default the data types only validate that the data is actually of that datatype, however you can add any validation you wish.

4.4.1 Base Validation Class

class AdvConfigMgr.config_validation.**ValidationsBase**

This is the base object that all other validation objects shoudl be based on. it is pretty simple at this point and is mainly a framework for consistency.

validate(*data*)

This is the main method for validation. This is called by the configuration manager and the data is passed to it. it should return that same data if it is validated, or raise an error or warning if not.

Raising an error will stop the processing, raising a warning will simply log the problem, and the developer can choose to poll the error queue and display the errors.

Parameters **data** –

Returns

4.4.2 Pre-Configured Validation Classes

class AdvConfigMgr.config_validation.**ValidationsBase**

This is the base object that all other validation objects shoudl be based on. it is pretty simple at this point and is mainly a framework for consistency.

validate(*data*)

This is the main method for validation. This is called by the configuration manager and the data is passed to it. it should return that same data if it is validated, or raise an error or warning if not.

Raising an error will stop the processing, raising a warning will simply log the problem, and the developer can choose to poll the error queue and display the errors.

Parameters **data** –

Returns

4.5 Migration Assist Classes

The system is designed to handle configuration changes during upgrades or downgrades. The following classes are used for this process. See Migration of Versions for information on how to configure and use migrations.

4.5.1 Base Migration Manager Class

class AdvConfigMgr.config_migrate.**ConfigMigrationManager**(*section*, **migration_dictionaries*)

set_migration(*version*)

Will set the version manager to use based on the database version. returns True if a version migration is found, False if no migration is found.

4.5.2 Base Migrations Actions Class

class AdvConfigMgr.config_migrate.**BaseMigrationActions**(*migration_manager*)

To add new migration / conversion actions, subclass this and add new methods, each method MUST accept as the first two args "value" and "option_name", after that, you can use whatever, though make sure to be consistent with how you define things in the migration dictionary.

You must also return a tuple of new_option_name, new_value, or None, None.

If your migration requires removeing an option, call `BaseMigrationActions._remove()` and it it requires adding a new option, call `BaseMigrationActions._new()`.

CHAPTER 5

Extending the system

License

```
GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc., <http://fsf.org/>
 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain
```

```
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.

    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.
```

```
    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
```

```
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
```

implemented by public license practices. Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded. In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time. Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number. If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation. If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission. For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
make exceptions for this. Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

                            NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER

```
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    {description}
    Copyright (C) {year}  {fullname}

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  {signature of Ty Coon}, 1 April 1989
  Ty Coon, President of Vice
```

```
This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

CHAPTER 8

Build / Test Status

From Travis-CI.org (this is probably red since it is testing all versions of Python against it. sorry, it only works with 3.4 for now).

a

A

B

C

D

F

G

I

L

O

R

S

T

V

W