# gs.config Documentation

*Release 2.2.0*

**GroupServer.org**

**Sep 27, 2017**

# Contents

This package provides some classes that are used to manage “sets” of configuration options.

Contents:

CHAPTER 1

# gs.config API Reference

The package exports the following API symbols.

## Configuration

**class** gs.config.**Config**(*configset*, *configpath=None*)

The configuration.

**Method** Config(configset, configpath=None)

**Parameters**

- **configset** (*str*) – The name of the configuration set to read.
- **configpath** (*str*) – The path to the configration file. If None *and* Zope is being used then etc/gsconfig.ini is read from instance directory.

**Raises**

- ***ConfigPathError*** – No path could be found.
- ***ConfigFileError*** – The configration file could not be read.
- ***ConfigSetError*** – The configuration file does not contain configset.

The actual parsing of the configuration file is done by the ConfigParser module.

**get**(*configtype*, *strict=True*)

Get the values defined in a section

**Parameters**

- **configtype** (*str*) – The ID for the section to retrieve.
- **strict** (*bool*) – If True (the default) then a ConfigNoOption error is raised when an option is present in the configuration file but absent from the schema. When False the option is ignored.

**Returns** The values for all options in the provided section.

**Return type** A `dict` containing `optionId: value` pairs. The values are coerced using the schema set by `set_schema()`.

**Raises**

- ***ConfigNoSectionError*** – No section for the ID in `configtype` exists.
- **ConfigNoOption** – An option was present in the configuration file but absent from the section.
- ***ConfigConvertError*** – An option could not be coerced.

**Example**:

```
smtpConfig = config.get('smtp')
```

**get_schema**(*configtype*)
Get the schema that is currently set for a section.

**Parameters** **confgitype** (*str*) – The identifier for the section.

**Returns** The schema that is currently set for the section.

**Return type** A `dict`, containing `optionId: type` pairs.

**Raises** ***ConfigNoSchemaError*** – No schema with the identifer `configtype` found.

**keys**()
Get the list of sections that are currently defined.

**Returns** A list of sections for the configuration set.

**Return type** `list`

**set_schema**(*configtype*, *schema*)
Set the schema that is used for parsing the options.

**Parameters**

- **configtype** (*str*) – The identifier for the section of the configration.
- **schema** (*dict*) – The schema for the section, as `optionId: type` pars.

When the value for an option in a section is retrieved its *type* is coerced from a string to one of the types passed in as **:param:'schema'**.

**Example**:

```
s = {'station': str,
     'frequency': int,}
conf.set_schema('radio', s)
```

`gs.config.`**getInstanceId**()
Get the ID of the current instance.

**Returns** The ID the of the instance, or `default`

**Return type** `str`

It can be useful to have multiple *instances* running on one server, but configured seperately. The `getInstanceId` function gets the ID of the current instance by looking it up in the `HTTP_INSTANCEID` property of the current Zope HTTP request. If Zope (and HTTP) are not being used then `default` is returned.

This function is defined in `gs.config` mostly out of convinience, as GroupServer normally organises its configuration into *instances*.

## Errors

The possible configuration errors.

**exception** `gs.config.errors.`**`ConfigConvertError`**
An error raissed when the value cannot be converted.

**exception** `gs.config.errors.`**`ConfigError`**
A generic error with the configuration.

**exception** `gs.config.errors.`**`ConfigFileError`**
An error with reading the config file

**exception** `gs.config.errors.`**`ConfigNoOptionError`**
An error raised when an option is not defined in a schema

**exception** `gs.config.errors.`**`ConfigNoSchemaError`**
An error raised when there is no schema

**exception** `gs.config.errors.`**`ConfigNoSectionError`**
An error raised when there is no configuration section specified.

**exception** `gs.config.errors.`**`ConfigPathError`**
An error with the path to the config file.

**exception** `gs.config.errors.`**`ConfigSetError`**
An error with the structure of the config file.

CHAPTER 2

# Example

Below is an example of a *file* with configuration sets, and *parsing* the file.

## File

In the following example the default configuration-set contains two items, `smtp` and `pop`. The the configuration for this set is provided by the `smtp-local` and `pop-local` sections. The `smtp-remote` section is currently unused by the configuration set.

```
[config-default]
smtp = local
pop = local

[smtp-local]
server = localhost
port = 2525

[pop-local]
server = localhost
port = 110

[smtp-remote]
server = smtp.example.com
port = 25
```

## Parsing

A configuration class is initialised. The second parameter is optional, depending on the degree to which we want the environment to configure things automatically.

```
>>> from gs.config import Config
>>> config = Config('default', '/example/file.ini')
```

A schema must be provided before data is retrieved. For example, setting the server to be a string, and a port to be an integer.

```
>>> config.set_schema('smtp', {'server': str, 'port': int})
```

Then a specific configuration section, with all the options can be retrieved as a dict.

```
>>> config.get('smtp')
{'port': 2525, 'server': localhost}
```

If file fails to fit the schema then a ConfigError is raised:

```
>>> config.set_schema('smtp', {'someparam': int})
>>> config.get('smtp')
```

```
Traceback (most recent call last):
  File "<console>", line 1, in <module>
  File "config.py", line 201, in get
    raise ConfigNoOptionError(msg)
ConfigNoOptionError: No option "server" defined in schema for "smtp".
```

However, it is possible to parse the configuration in a lax way, by passing `strict=False`. This allows for hard-coded defaults:

```
>>> config.get('smtp', strict=False)
{}
```

CHAPTER 3

# Changelog

## 2.2.0 (2015-03-17)

- Allowing the parsing of the configuration file to be lax

## 2.1.3 (2014-09-16)

- Renaming the `*txt` files as `*rst` files, for GitHub

## 2.1.2 (2014-06-24)

- Fixing a coding error, where a `m` was used in an error, rather than a `msg`

## 2.1.1 (2014-05-05)

- Added Sphinx documentation

## 2.1.0 (2014-04-25)

- Adding Python 3 support
- Updating documentation
- Adding unit tests and tox support

## 2.0.0 (2013-08-10)

- Merge of some code from `gs.database`
- Reduced dependency on Zope (now an extra requirement)

## 1.0.0 (2012-07-13)

- Initial version

CHAPTER 4

# Resources

- Documentation: http://groupserver.readthedocs.org/projects/gsconfig
- Code repository: https://github.com/groupserver/gs.config
- Questions and comments to http://groupserver.org/groups/development
- Report bugs at https://redmine.iopen.net/projects/groupserver

CHAPTER 5

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## g

# Index