Asterisk IVR Documentation

Release 0.0.2

Brian LaVallee

Mar 15, 2019

Contents

Contents:

CHAPTER 1

Basic Settings

CHAPTER 2

Database Connection Settings

2.1 Asterisk Configuration File

This module is used to read the database settings from an Asterisk configuration file. It uses the Python ConfigParser to load the setting values directly from the specified section (context) of the Asterisk configuration file.

Note: You need the MySQL Connector/Python installed on your system to connect to the database.

Example

Import the module, get the config settings, and pass them to your mysql connection:

```
import ivr.connection

config = ivr.connection.config()
 -or-
config = ivr.connection.config('context')

database = mysql.connect(**config)
```

2.2 Configuration

`connection.`**`asterisk_path = '/etc/asterisk'`**

String – Asterisk Configuration Directory Path

The path to the Asterisk Configuration Files for **your** installation.

`connection.`**`asterisk_conf = 'res_config_mysql.conf'`**

String – Name of the configuration file you want to use.

> **Warning:** The MySQL Connector/Python does not support DSN configuration options.
>
> **Do not use Asterisk ODBC configuration files!**

You can use any Asterisk configuration file that contains the host, database, user, and password details. *You could create a dedicate configuration file too.*

`connection.`**`context = 'general'`**

String – The section (context) of the configuration file to use.

The Asterisk configuration file can be read by the Python ConfigParser. Just specify the section of the configuration you would like to use. The section is contained in square brackets, as shown in the following configuration file example:

```
[general]
dbhost = 127.0.0.1
dbname = asterisk
dbuser = myuser
dbpass = mypass
dbport = 3306
dbsock = /tmp/mysql.sock
dbcharset = utf8
requirements=warn  ; or createclose or createchar
```

2.3 Connection Settings

`connection.`**`config = <function config>`**

CHAPTER 3

Read the Docs

The real **power** of Read the Docs is the ability to generate documentation from the docstrings contained within your code. *The only problem is how to generate this documentation, isn't documented well.*

3.1 Banging on the Keyboard

Getting it working, getting Read the Docs to generate documentation using the docstrings was a frustrating experience. Lot's of *Banging on the Keyboard*, trying different things, and making changes all over the place.

Installing `sphinx` locally and running `sphinx-quickstart` **may** have been the solution.

..or helped get closer to making the documentation with `docstrings`.

3.2 Importing Python Docstrings

Dealing with external modules or system I/O, Read the Docs is **not** importing the docstrings. It's not showing any output from the module. See: Basic Settings which *should* import the docstrings shown below:

```
# -*- coding: utf-8 -*-
"""
A Simple Example
"""

from asterisk.agi import *

myVar = 'some data'
"""
A Simple Variable
"""

agi = AGI()
```

(continues on next page)

(continued from previous page)

```
"""
Create an AGI Instance
"""

agi.answer()
"""
I/O Communications: stdin, stdout, and stderr
"""
```

Warning: This does not work. Read the Docs doesn't display anything.

It appears to choke when it reaches `agi = AGI()` unless **Install Project** is enabled in Read the Docs.

This happens because `AGI()` calls an external module.

Warning: `agi.answer()` does not work, with or without Read the Docs installing the project inside a virtualenv.

It appears to choke when it reaches `agi.answer()`

The is probably due to the `answer()` function working with `stdin` and `stdout`.

3.2.1 Mock

It appears there **may** be a way to `mock` the module function. Telling Read the Docs that it **should not** test the function. But I have not been able to do this successfully.

CHAPTER 4

Stub

4.1 Sample H3

4.1.1 Sample H4

Sample H5

Sample H6

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

Index