Sidewalk Documentation

Release 0.3.0

Blake Rohde

January 19, 2013

CONTENTS

Sidewalk allows you to easily trigger the execution of sets of Python methods from the command-line.

Simply register your activity processors (i.e. functions/methods) with key associations and execute them by key or group using the included command-line utility `sidewalk`.

This comes in handy when you want to setup cron jobs to execute Python code. You can easily create multiple cron job entries that run at various times and call different sets of activity processors.

CHAPTER

ONE

USER GUIDE

This section of the documentation provides the user of Sidewalk a guide for how to use the command-line interface through the `sidewalk` executable.

1.1 Introduction

1.1.1 Why Sidewalk?

Sidewalk allows you to trigger the execution of Python methods from the command-line.

As the saying goes, there are many ways to skin a cat. Obviously people have been executing python code one way or another via command-line through methods of "python -c", or pointing directly to a Python file and using the "'__name__' == '__main__'" convention, etc.

When you register an activity processor, you assign it a key. You can create group associations when assigning keys (e.g. 'group.sub-group.activity_processor'). You can then execute all of the registered activity processors in one call by referencing the assigned 'group' or 'group.sub-group'. This is a nice feature as it allows you to execute many activity processors, that are potentially located in any number of modules, in one call.

You can also execute any number of groups, and/or any number of specific activity processors, when calling the command-line utility `sidewalk`. This is a key feature of Sidewalk, as you can easily create multiple cron job entries that run at various times, but call different activity processors. If you wanted to implement similar functionality in a traditional sense, you would have to create multiple modules that reference said activity processors, or create one module and have an if-else statement to do the same.

1.1.2 Sidewalk's License

1.2 Setup

To setup Sidewalk on your system, please follow the Installation and Run sections below.

1.2.1 Installation

Install Sidewalk via pip:

```
$ pip install sidewalk
```

If pip is not available on your system, you can also install via easy_install:

```
$ easy_install sidewalk
```

If PyPi is down or you otherwise cannot use it, you can also install Sidewalk with pip using the GitHub-hosted tarball:

```
$ pip install -Iv https://github.com/blakerohde/sidewalk/tarball/master
```

To install without pip or PyPi, download the source tarball and run the following in the un-tarred directory:

```
$ python setup.py install
```

1.2.2 Run

Now that Sidewalk is installed on your system, you can verify the command-line utility is installed by executing the following:

```
$ sidewalk --help
```

1.3 How to use Sidewalk

This section of the documentation provides commentary for how to use Sidewalk.

This page of documentation will be expanded in the future. In the mean time, here is a quick run-down for how to get started:

1.3.1 Create Activity Processors

- This can be any function defined in any module.
- It is common to create an `activitiy_processors/` directory in your project's main directory to hold all of the activity processors specific to that project. This directory is also typically used to the project's Sidewalk configuration file (see Step 1 below).
- Note Sidewalk comes with an example activitiy processor `hello` in the module `sidewalk.test.example`.

1.3.2 Create A Configuration File

Before we can register our activity processors, we need to create and initialize a configurationfile. This file will contain a list of registered activity processors. It is up to you where to put the configuration file. You can create a single configuration file for an entire system, or create one for each of your projects. For right now, lets just create one in the current working directory:

```
$ sidewalk ./sidewalk.conf init
```

1.3.3 Register Activity Processors

```
$ sidewalk ./sidewalk.conf add example.hello sidewalk.test.example.hello
```

- The `sidewalk` command-line utility is your way to easily add, list/view, and remove activity processors.
- Note that when adding your activity processors, you are creating a key association to the activity processor. Also note the syntax: `example.hello`, in this case, `example` is the group and `hello` is the name. Groups are useful for executing multiple activity processors in one go. See step 3 below for more information.

1.3.4 Execute Activity Processors

```
$ sidewalk ./sidewalk.conf pave example.hello
```

- Here we are executing our activity processor `example.hello`.
- You can select activity processors by group, e.g.: `$ sidewalk ./sidewalk.conf pave example.`.
- You can execute any number of activity processors in one call, e.g.: `$ sidewalk ./sidewalk.conf pave example.` to execute all activity processors of group `example` or `$ sidewalk ./sidewalk.conf pave example.hello second_group.` to execute `example.hello` and all activity processors of group `second_group`.

CHAPTER

TWO

API DOCUMENTATION

This section provides a technical overview of the underlying logic that drives Sidewalk.

2.1 sidewalk Package

2.1.1 `sidewalk` Package

sidewalk

copyright

3. 2013 by Blake Rohde.

license ISC, see LICENSE for more details.

2.1.2 Subpackages

conf Package

`global_settings` Module

core Package

`activity_aggregator` Module

sidewalk.activity_aggregator

This module is the primary workhorse powering Sidewalk. Let's execute some activity processors!

copyright

3. 2013 by Blake Rohde.

license ISC, see LICENSE for more details.

class `sidewalk.core.activity_aggregator.`**`ActivityAggregator`**(*filename=None*, *active_activity_processor_keys=[]*, *active_group_keys=[]*)

The `ActivityAggregator` object. It allows for the execution of the defined activity processors.

import_module(*name*)
Simple wrapper for importing the requested module.

run()
The primary workhorse. Runs/executes the active activity processors.

exceptions Module

sidewalk.exceptions

This module contains custom exceptions that can be thrown by Sidewalk.

copyright

3. 2013 by Blake Rohde.

license ISC, see LICENSE for more details.

exception sidewalk.core.exceptions.**SidewalkGroupDoesNotExist**(*group_key*)
Bases: exceptions.Exception

Activity processor group requested is not defined.

exception sidewalk.core.exceptions.**SidewalkKeyDoesNotExist**(*key*)
Bases: exceptions.Exception

Activity processor requested is not defined.

exception sidewalk.core.exceptions.**SidewalkMethodDoesNotExist**(*module*, *method*)
Bases: exceptions.Exception

The Activity processor (method) does exist in the specified module.

exception sidewalk.core.exceptions.**SidewalkModuleImportError**(*module*)
Bases: exceptions.Exception

Activity processor module could not be imported.

exception sidewalk.core.exceptions.**SidewalkRogueActivityProcessor**(*activity_processor*)
Bases: exceptions.Exception

The Activity processor threw an unhandled exception.

exception sidewalk.core.exceptions.**SidewalkSectionNotDefined**(*filename*, *section*)
Bases: exceptions.Exception

The specified settings file does not contain a required section.

exception sidewalk.core.exceptions.**SidewalkSettingsFileIOError**(*filename*, *permission*)
Bases: exceptions.Exception

Settings file IOError.

loggers Module

sidewalk.loggers

This module contains simple loggers for printing pretty messages.

copyright

3. 2013 by Blake Rohde.

license ISC, see LICENSE for more details.

sidewalk.core.loggers.**debug**(*message*, *verbose=True*)
Debug log wrapper for log.

sidewalk.core.loggers.**error**(*message*, *verbose=True*)
Error log wrapper for log.

sidewalk.core.loggers.**log**(*message*, *tag='GENERAL'*, *verbose=False*)
Simple log/message function for the output of timestamped messages.

manager Module

sidewalk.manager

This module contains manager(s) for managing Sidewalk resources (setting files, etc.).

copyright

3. 2013 by Blake Rohde.

license ISC, see LICENSE for more details.

class sidewalk.core.manager.**ActivityProcessorsManager**(*filename=None*)
The ActivityProcessorsManager object. It allows for the management and storage of activity processors defined in the primary settings.conf config file.

add(*key*, *activity_processor*)
Add a new activity processor.

get_activity_processor(*key*)
Get the specified activity processor by its key.

get_activity_processor_pairs(*key_list=None*)
Get the specified activity processor pairs. Pairs are returned as a dictionary where the key of the dict is the activity processor's key and the dict's associated value is the activity processor's path.

If no key list is specified, all of the defiend activity processor pairs are returned.

get_group_activity_processor_pairs(*group_key*)
Get the specified activity processor pairs with the specified group key.

open(*filename*)
Open the specified configuration file.

remove(*key*)
Remove the specified activity processor by its key.

remove_group(*group_key*)
Remove the specified activity processors by their group key.

save()
Save the changes made.

sidewalk.core.manager.**get_conf**(*path*)
Get the path of the requested configuration file.

test Package

example Module

sidewarlk.test.example

This module contains sample activity processors.

sidewalk.test.example.**baddie_unhandled_test**()
A test activity processor that throws an uncaught exception.

sidewalk.test.example.**hello**()
Simply print hello world back to the console.

sidewalk.test.example.**import_test**()
Import the core sidewalk module to check against a defined value.

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

S