AutoAnt Documentation

Release 0.2.0

Daniel Vaz Gaspar

June 26, 2016

1 Fixes, bugs and contributions 3
2 Installation 5
3 Using pip 7
4 The Idea 9
5 Quick HowTo 11
6 Producers and Processors 13
7 Producer - Directory Monitor 15
8 Processors 17
9 Processor - SCP 19
10 Processor - SMB 21
11 Processor - FTP 23
12 Processor - Rename 25
13 Processor - Move 27
14 Processor - Copy 29

AutoAnt is an object processing automation package. It's ideal for file processing, if you need to process/syncronize/move files on several directories to different locations (local or remote) and then renaming/moving/deleting or what ever you want. This is the answer.

It's extremely flexible and extensible, just describe your Sys Admin file/*something* processing nightmare on a JSON file.

CHAPTER 1

Fixes, bugs and contributions

You're welcome to report bugs, propose new features, or even better contribute to this project.

Issues, bugs and new features

Contribute

CHAPTER 2

Installation

Installation is straightforward, using the normal python package install.

CHAPTER 3

Using pip

- **Simple Install**

  You can install AutoAnt simply by:

```
$ pip install autoant
```

- **Advised Virtual Environment Install**

  Virtual env is highly advisable because the more projects you have, the more likely it is that you will be working with different versions of Python itself, or at least different versions of Python libraries. Let's face it: quite often libraries break backwards compatibility, and it's unlikely that any serious application will have zero dependencies. So what do you do if two or more of your projects have conflicting dependencies?

  If you are on Mac OS X or Linux, chances are that one of the following two commands will work for you:

```
$ sudo easy_install virtualenv
```

  or even better:

```
$ sudo pip install virtualenv
```

  One of these will probably install virtualenv on your system. Maybe it's even in your package manager. If you use a debian system (like Ubuntu), try:

```
$ sudo apt-get install python-virtualenv
```

  Once you have virtualenv installed, :

```
$ mkdir myproject
$ cd myproject
$ virtualenv venv
New python executable in venv/bin/python
Installing distribute............done.
$ . venv/bin/activate
(venv)$
```

  Now install **AutoAnt** on the virtual env, it will install all the dependencies and these will be isolated from your system's python packages

```
(venv)$ pip install autoant
```

  Next you can run the command line utility, to check if everything is ok :)

```
        (venv)$ autoant_console -h
```

CHAPTER 4

The Idea

AutoAnt structure is based on **Producers** and **Processors**, every item produced is subject to a processing sequence. You can have many **Producers** associated to Nth processing sequences.

Producers - Will produce objects to be processed.

Processors - Will process the objects.

AutoAnt has a command line utility that you can use for easy scheduling, or on your own scripts. It will read a JSON config file that describes the automation process.

The config file is a list of objects containing the following structure:

```
[
    {
        "producer_sequence": [
            {
                "name": "SOME UNIQUE ID",
                "type_key": "KEY OF PRODUCER",
                ...
            },
            ...
        ]
        "processor_sequence": [
            {
                "name": "SOME UNIQUE ID",
                "type_key": "KEY OF PROCESSOR",
                ...
            },
            ....
        ]
    },
    ....
]
```

It's config is a list of automation *steps* that can be run on a different threads (optional). Each *step* is an object with a list of producers and a list of processors. Every *item* (files, lines on a file, db records) is submitted to the processing sequence.

Every producer and processor has a type and a name, this is required.

CHAPTER 5

Quick HowTo

Enough talk, let's go right into a quick example.

Let's say you have some file processing to do, and you need to automate it, probably you have already made tons of similar scripts on file automation, but every time you have a new problem you have to write something new, for the same abstract issue.

On our example, you have a database that generates data files, these files are supposed to be processed on a remote server, for the first task you have to copy every new file to a remote FTP server.

We are going to write a JSON configuration file describing AutoAnt solution:

```
[
    {
        "producer_sequence": [
            {
                "name": "DBSOURCE",
                "type_key": dir_mon",
                "basedir": "/db/export/contacts",
                "recursive": "True"
            }
        ]
        "process_sequence": [
            {
                "name": "Remote",
                "type_key": "ftp",
                "remote_dir": "/contacts",
                "remote_host": "remoteserver.domain.com",
                "username": "user",
                "password": "password"
            }
        ]
    }
]
```

Know add to your scheduling system **crontab** on UNIX or **Scheduled Tasks** on Windows.

crontab:

```
*/5 * * * * autoant_console --config /home/of/config/config.json &>> autoant.log
```

Note: If you're running on a python's virtual env has advised, you will have to write a small script to activate the enviroment and then execute autoant.

AutoAnt will every 5 minutes look for new files on your local directory */db/exports/contacts/ every new file will be sent to *remoteserver.domain.com* . This is ok, what will AutoAnt add to this apparently simple task

- You will have a detailed and highly configurable log, using python's standard lib, logging.
- If something goes wrong on your file processing (remote server is down or something), the failed files will be reprocessed next time, without the use of moving/coping/renaming the succeeded ones.
- The copy is recursive and differential the directory structure will be created on the remote site.
- If a file is still open (being created by the database on this example), the file is not processed this time. (Linux only feature).
- Integrated extensible highly configurable system.
- Over loop prevention, AutoAnt will not run if another instance using the same config is still processing.

Now your company wants to copy the same files to a different location but this time they only accept SFTP (they probably know better then FTP). Just add a json object to the 'process_sequence' property:

```
[
    {
        "producer_sequence": [
            {
                "name": "DBSOURCE",
                "type_key": dir_mon",
                "basedir": "/db/export/contacts",
                "recursive": "True"
            }
        ]
        "process_sequence": [
            {
                "name": "Remote",
                "type_key": "ftp",
                "remote_dir": "/contacts",
                "remote_host": "remoteserver.domain.com",
                "username": "user",
                "password": "password"
            },
            {
                "name": "Remote2",
                "type_key": "scp",
                "remote_dir": "",
                "remote_host": "remoteserver2.domain.com",
                "username": "user2",
                "password": "password"
            }
        ]
    }
]
```

You have two remote copies, and if either fails they will be reprocessed. If a file put or connection fails, on **Remote2** second copy, it will be resent next up time and only to *remoteserver2.domain.com*.

Remember each item on a processing sequence is independent by default.

If you want to make them dependent on the success or failure of the previous processor, just add the 'dependent' property with value 'True' to the processor. This way all failed items will not be processed by the next processor, this is useful for many purposes like coping files and renaming them, if a copy fails the file will not be renamed.

Note that the name property is a free tag, use it for giving a friendly name for your directory monitoring and processing tasks. Make sure they are unique on their JSON structure. They will be used to create info files from AutoAnt named on this example: DBCONTACTS.Remote.sav and DBCONTACTS.Remote2.sav. and your log file will have this tags on each line.

Producers and Processors

All producers and processors share the following properties:

Key	Description
name	A unique user's free tag to id the producer
type_key	the type of producer, run autoant_console -p to list all available

All producers share the following properties

Key	Description
thread	Will run the producing process and it's associated processor on a separate thread.

All processors share the following properties

Key	Description
dependent	A boolean property 'True'/'False' to make a processor dependent of the preceding processor success

CHAPTER 7

Producer - Directory Monitor

This producer key is **"dir_mon"**. And produces *FileItem* objects.

This producer will scan recursively or not a local directory and collects all files to be processed

Configuration properties are:

Key	Description
basedir	The local directory to monitor
recursive	(Optional) boolean string (True/False) collects file recusively or not. Default is True
filter	(Optional) regular expression, all files must pass.
mtime	(Optional) Modification time stamp in minutes. If negative ex: -5 will produce files modified less then 5 minutes ago. **If Positive ex: 5 will produce files modified more then 5 minutes** ago.
atime	(Optional) Same has mtime but for accessed time stamp.
ctime	(Optional) Same has mtime but for created time stamp.

Example: If you want to monitor only text files from a directory use:

```
{
    "name": "AUTOANT",
    "type_key": "dir_mon",
    "basedir": "/home/dpgaspar/workspace/autoant/",
    "filter": ".*.txt$"
}
```

CHAPTER 8

Processors

All processors share a common property named state that can be True or False. By default it's enabled. When enabled, will not process items that were already processed on a previous run. If turned to False, it will always process everything, every time.

Key	Description
state	Keeps state between runs. will record successfully processed items

Processor - SCP

This processor key is **"scp"**

Will **Put** files remotely using SFTP protocol. Reconstructs missing directories structure. This processor needs python's excellent **Paramiko** package.

Their configuration properties are:

Key	Description
remote_dir	The remote directory where files will be copied to
remote_host	Remote host IP or network name.
username	The username for authentication
password	(Optional) The password for authentication
key_filename	(Optional) The key RSA file for authentication
timeout	(Optional) The connection's timeout.
channel_timeout	(Optional) The channel timeout.

CHAPTER 10

Processor - SMB

This processor key is **“smb”**

Will **Copy** files remotely using SMB protocol (Windows file share). Reconstructs missing directories structure. This processor needs python’s excellent **pysmb** package.

Their configuration properties are:

Key	Description
remote_dir	The remote directory where files will be copied to
remote_host	Remote host IP or network name.
remote_name	The NETBIOS remote computer name.
local_name	THE NETBIOS local computer name.
username	The username for authentication
password	(Optional) The password for authentication
timeout	(Optional) The connection’s timeout.

Processor - FTP

This processor key is **“ftp”**

Will **Put** files remotely using FTP or FTPS protocol. Reconstructs missing directories structure.

Their configuration properties are:

Key	Description
remote_dir	The remote directory where files will be copied to
remote_host	Remote host IP or network name.
remote_port	(Optional) Remote host port number for FTP. (default 21)
username	The username for authentication
password	(Optional) The password for authentication
is_ssl_auth	(Options) Encrypts authentication (True/False)
is_ssl_data	(Optional) Encrypts data (True/False)
timeout	(Optional) The connection’s timeout.
debug_level	(Optional) python’s ftplib debug level, 0,1 or 2.

CHAPTER 12

Processor - Rename

This processor key is **"rename"**

Will rename files using a python's regular expression.

Their configuration properties are:

Key	Description
rule_origin	Regular expression to capture all or part of the filename
rule_destination	Regular expression to transform the captured part from rule_origin

Example: Rename recursively all files with prefix "_":

```
{
    "name": "RENAME_1",
    "type_key": "rename",
    "rule_origin": "(.*)",
    "rule_destination": "_\\1"
}
```

Attention if you run this example many times it will add up "_" ahead of file names.

CHAPTER 13

Processor - Move

This processor key is **“move”**

Will move files to a different directory.

Their configuration properties are:

Key	Description
dest_dir	Destination directory

Example: Move files recursively:

```
{
    "name": "MOVE_1",
    "dest_dir": "/name/of/the/destination/dir"
}
```

Processor - Copy

This processor key is **“cp”**

Will copy local files to a different directory.

Their configuration properties are:

Key	Description
dest_dir	Destination directory

Example: Copies files recursively:

```
{
    "name": "COPY_1",
    "dest_dir": "/name/of/the/destination/dir"
}
```