PyFarm Documentation

Release latest

Oliver Palmer, Guido Winkelmann

February 08, 2017

Contents

Attention: These documents are still undergoing revision

PyFarm is a Python based distributed job system which is intended to be easy to deploy and maintain. Initially developed for individual use new revisions have been engineered with larger deployments in mind. It's a system that's focused on providing a framework for customized command execution while taking into account resource management.

Since this project is under active development, if you have specific questions or comments you're always welcome to post to the google groups discussion list or via email to pyfarm@googlegroups.com.

CHAPTER 1

Components

PyFarm has several distinct roles that it must fill in order to accomplish what it was designed to do. Because of this the project has been broken down into three components to help manage scope and limit interdependencies as much as possible.

1.1 Core

| | Documentation | Source Code

This is the base library which provides a few modules and objects which the other two components of PyFarm use. The primary purpose of this component is to centralize code basic functionality such as logging and configuration.

1.2 Master

| | Documentation | Source Code

This is the component responsible for storing jobs and tasks to run as well as allocation of work to remote hosts. This component is the largest of the three components and contains the code necessary to run the web interface, interact with the relational database, REST APIs and scheduler.

1.3 Agent

| | Documentation | Source Code

This component controls the execution of commands on remote systems as instructed by the master. This component also contains the job types which are used as a framework for executing commands.

Contents

1.3.1 Contributing To PyFarm

Summary

PyFarm is a Python based project with the goal of creating an easy to use and deploy render farm. The project is written on top of several existing libraries including SQLAlchemy, Flask, Twisted, and many more. While PyFarm's original goal is providing the base for running a render farm it can be used for other types of work as well.

As flexible and easy to use as the project may be, contributions from the community are always welcome. There are many ways one can contribute to the overall project but generally they fall under either documentation, bug fixes, features, or testing. With that in mind, if you're looking to contribute to any of these three areas then read on to get started.

Developer Resources

Below are some resources which are for developers wishing to contribute to PyFarm.

Coding Style

PyFarm developers should follow the conventions set fourth in **PEP 8** unless told to do so otherwise. Any exceptions or more specific example will be noted below and other may be added as time progresses. Coding style is an important part of PyFarm because it keeps the code consistent and readable which contributes to the overall quality of the source code.

One final thing to note is that there are places where there's inconsistencies in style and there always will be. It's up to the team to maintain the style so if you spot something inconsistent with this guide while working you should always feel free to fix it.

For other areas where there's contention about style the developers should come to a consensus and then add their findings to this document.

Whitespace Whitespace is an important component of any Python program. The below sets out the standards that PyFarm follows which in practice is not far off from **PEP 8**

Tabs and Spaces Follow PEP8 and use spaces. To be more exact when working on PyFarm please use four spaces. For non-python source code please also use four spaces unless the language you're working with explicitly does not allow this.

Basic Syntax Considerations As with the parts above you should follow **PEP 8** when considering how you use Python's basic syntax structures. Below are some additional considerations that are specific to PyFarm as well as some short explanations.

Quotations Python has two forms of quotations `'` and `"`. Functionally there's not any difference between two however all code, strings, error messages, etc should use `"`. The exception to this rule is if you need `"` inside of a string, then you should use `':'hello "world"'`. The reason for this rule is two parts:

- it's easy to be, and often is, inconsistent when you mix `'` and `"`
- developers from other languages, such as C++, are more more used to using `"` for strings instead of `'`

Standard Documentation Docstrings are high encouraged for all callable functions, methods, classmethods, and staticmethods. When creating a docstring please use `"""` instead of `'''` to enclose the documentation.

```
def foo():
    """This is a single line doc string"""

def bar():
    """
    This is a multi-line documentation string.  When you need to
    use multiple lines you should keep the left and right side of
```

```
    the opening and closing quotes clear.
    """
```

Endpoint Documentation A large part of PyFarm is `pyfarm.master` which includes HTTP endpoints serving as the master's API. It's important to document these using the `sphinxcontrib.httpdomain` syntax so it's readable. Take special note that in the top level url the type and name of the thing being posted is in the url, `<str:item>`, however in the examples it's the real text.

```
from flask.views import MethodView
class FooItemsAPI(MethodView):
    def post(self, item=None):
        """
        ``POST`` method which

        .. http:post:: /api/v1/foo/<str:item> HTTP/1.1

            **Request**

            .. sourcecode:: http

                POST /api/v1/foo/foobar HTTP/1.1
                Accept: application/json

                {
                    "item": "foobar"
                }

            **Response (new item created)**

            .. sourcecode:: http

                HTTP/1.1 201 CREATED
                Content-Type: application/json

                {
                    "item": "foobar",
                    "id": 1
                }

        :statuscode 200: an existing tag was found and returned
        :statuscode 201: a new tag was created
    """
```

Which ends up looking like this when rendered:

POST /api/v1/foo/<str:item> HTTP/1.1

Request

```
    POST /api/v1/foo/foobar HTTP/1.1
    Accept: application/json

    {
        "item": "foobar"
    }
```

Response (agent newly tagged)

```
    HTTP/1.1 201 CREATED
    Content-Type: application/json

    {
        "item": "foobar",
        "id": 1
    }
```

statuscode 200 an existing tag was found and returned

statuscode 201 a new tag was created

Line Continuations The default max line length for the project is 80 characters. Anything longer should use a line continuation if it can't be split up otherwise.

```
# import continuations should use (), it's cleaner and easier to
# modify later on
try:
    from httplib import (
        BAD_REQUEST, NOT_FOUND, UNAUTHORIZED, INTERNAL_SERVER_ERROR)
except ImportError:
    from http.client import (
        BAD_REQUEST, NOT_FOUND, UNAUTHORIZED, INTERNAL_SERVER_ERROR)

# preferred
message = ("this is a message which you may not be "
           "able to fit onto one linet")

# but this is ok too
message = "this is a message which you may not be " \
          "able to fit onto one line"

# preferred
if (a == b and c == d and a == b
        or a and b and c and d):
    pass

# but this is ok too
if a == b and c == d and a == b \
        or a and b and c and d:
    pass
```

HTTP Endpoints

URL Formatting The following rules should be applied when constructing an HTTP endpoint:

- endpoints referring to objects should be plural so `/items/` instead of `/item/`
- any endpoint that's not referring to a specific document should contain a trailing slash: `/items/`
- endpoints that refer to a specific document shouldn't contain a trailing slash `/items/1`
- when working with groups of items under a single item the trailing slash should be added `/items/1/children/`
- any endpoint that's an API should contain a version number `/api/v1/items/`

Validating Data in API Endpoints Most of the time you'll want a standard way of validating the incoming request before you have to deal with it yourself. For this there's the `validate_with_model` function that in combination with `before_request` will:

- ensure the incoming data to the API is json
- test the incoming data to ensure it has all the required keys
- test to make sure the incoming data does not contain keys that don't exist in the table
- check to ensure that all data included matches the expected types based on the types in the model
- set `flask.g.json` if all of the above proceed without problems
- return a useful error message in response to the request if there's problems

A short example of how this works is below

```
try:
    from httplib import CREATED
except ImportError:  # pragma: no cover
    from http.client import CREATED

from flask import g
from pyfarm.master.application import app, db
from pyfarm.master.utility import validate_with_model, jsonify
from pyfarm.models.tag import Tag

# NOTE: this is an example only, not functional code as it does not
# setup the route
@validate_with_model(Tag)  # does all the validation in the points above
def put_tag():
    model = Tag(**g.json)
    db.session.add(model)
    db.session.commit()
    return jsonify(model.to_dict()), CREATED
```

Logging

General You are welcome to use the print function on your own but before pushing code or writing tests please switch to a logger:

```
from pyfarm.core.logger import getLogger
logger = getLogger("foobar")
```

The above will create a logger under the proper namespace with a reasonable set of defaults applied. It will also create it under the proper namespace, in this case `pyfarm.foobar`.

> **Warning:** The above is not actually true for the agent and job types. Those will require a special logging setup which is not yet addressed in this guide.

Usage Below are some general guidelines that apply specifically to logging to minimize potential performance problems and decrease inconsistencies in usage. The following examples assume the code in the section above was run.

Log Formatting When providing arguments to the logger use lazy formatting

```
greeting = "morning"
logger.info("good %s", greeting)
```

Use %r For Objects Instead of repr() Instead of always calling repr() on the object just use the %r string formatter

```
data = {"true": True, "none", None}
logger.info("data: %r", data)
```

Exceptions and Errors At some point you'll have to handle or produce exception within PyFarm. Depending on where in the code base you're working the patterns may vary so please see below for more information.

Suppressing All Exceptions Always use `try: except Exception` when you must suppress all unhandled exceptions. It's also advised that you log the original exception message too so we can find and better handle these errors in the future.

```
try:
    foobar()

# always document exactly why you're suppressing
# all unhandled exceptions.  Generally speaking there
# are **very few** cases where this should ever be a standard
# practice.
except Exception as e:
    logger.exception(e)  # this is sometimes skipped
    logger.warning("unhandled exception: %s", e)
    pass
```

Custom Exceptions PyFarm used to throw more custom exceptions but since then nearly all of the code has switched back to using standard exceptions. In the event a custom exception must be created it should follow the general pattern below.

```
class PyFarmBaseException(Exception):
    """The base exception which all PyFarm exceptions derive from"""
    pass

# you may optional subclass from a related builtin type too
class FileHandlingError(PyFarmBaseException):
    """Raised when there's a problem handling files"""
    pass
```

Throwing Exceptions Inside A Request When working with `pyfarm.master` you'll often need to throw exceptions that will be used as responses to a request. There's a couple of ways to do this:

Default Method This is the standard method for throwing exceptions in the web application in response to a request. The below code will cause `pyfarm.master.errors.error_400()` to produce an error response to the request depending on the mimietype. For example if the incoming request it `application/json` the below will construct a json response.

```
try:
    from httplib import BAD_REQUEST
except ImportError:  # pragma: no cover
    from http.client import  BAD_REQUEST

from flask import g, abort
from pyfarm.master.application import app

@app.route("/foobar/")
def foobar():
    # NOTE: like logging incomplete or single sentences should
    # start with a lower case letter
    g.error = "something went wrong"
    abort(BAD_REQUEST)
```

Alternate Method Although uncommon in other cases it may make sense to response directly when there's a problem.

```
try:
    from httplib import BAD_REQUEST
except ImportError:  # pragma: no cover
    from http.client import  BAD_REQUEST

from flask import g, abort
from pyfarm.master.application import app
from pyfarm.master.utilities import jsonify

@app.route("/foobar/")
def foobar():
    # NOTE: like logging incomplete or single sentences should
    # start with a lower case letter
    return jsonify(error="something went wrong"), BAD_REQUEST
```

Platform Specific Code PyFarm is a cross-platform application and because of this some consideration about support multi-platforms in the same code base must be considered.

Import Handling Imports for platform specific modules should be setup like below. This is better than simply `except ImportError: pass` because the exception thrown in the event of misuse will make more sense. In cases where you've tried the best you can to determine the proper coarse of action raise an exception that describes the situation best.

```
try:
    from os import fork
except ImportError:
    fork = NotImplemented

try:
    import win32process
except ImportError:
    win32process = NotImplemented

if fork is NotImplemented and win32process is not NotImplemented:
    subprocess.Popen(
    commands, creationflags=win32process.DETACHED_PROCESS)
```

```
elif fork is not NotImplemented:
    os.fork()

else:
    raise NotImplemented(
        "failed to determine correct way to launch process")
```

Internal Logic Handling If you're not working with imports like above and you just need to know what platform you're on use constants from `pyfarm.core.enums`.

```
from pyfarm.core.enums import (
    LINUX, MAC, WINDOWS, POSIX, CASE_SENSITIVE_ENVIRONMENT,
    CASE_SENSITIVE_ENVIRONMENT)
```

Supporting Multiple Python Versions PyFarm supports Python 2.6+ in most modules except for `pyfarm.agent` or `pyfarm.jobtypes` which currently supports only Python 2.6 and Python 2.7 due to problems with Twisted and Python 3.x. Because of this certain considerations must be made when working on the project.

Checking Python Versions `pyfarm.core.enums` has some special constants for getting the current Python version. There are other ways of checking the Python version however these constants are provided for consistency and readability.

```
from pyfarm.core.enums import PY26, PY26, PY2, PY3
```

Type Information Certain types consolidated or removed when Python 3 was released. Because of this some of the older ways of checking for basic types had to change. Again `pyfarm.core.enums` should be used for consistent and clean behavior across Python versions.

```
from pyfarm.core.enums import STRING_TYPES, NUMERIC_TYPES
```

2.x vs. 3.x Version Specific Python Imports Certain built-in imports where also consolidated or renamed when Python 3 came about. Rather than using constants to do a version check let Python's import system do the work for you.

```
# Python 2.x imports should always go first since
# most studios and operating systems that ship with Python
# still default to 2.x
try:
    from UserDict import UserDict
except ImportError:  # pragma: no cover
    from collections import UserDict

# for objects or functions that were renamed
try:
    _range = xrange
except NameError:  # pragma: no cover
    _range = range

# for attributes which have changed
data = {}
try:
    items = data.iteritems
```

```
except AttributeError:
    items = data.items
```

Backwards Compatible Imports Sometimes you'll need access to new functions or modules that don't with whatever Python version or package you're working with. In these situations, like with version specific Python imports, you should use the import system to make the decision for you.

```
# NOTE: Python 2.6 and up includes json, which is what PyFarm requires,
# this is just an example
try:
    import json
except ImportError:  # pragma: no cover
    import simplejson as json
```

Project Structure

Sub-Projects

The project is broken down into several smaller sub-projects to aid in long term maintenance and isolation of code scope. Generally speaking there are two kinds of sub-projects, supporting and operational. `Supporting` sub-projects support all consumers of the project in some capacity (ex. documentation or deployment tools). `Operations` sub-projects contain the code which operate PyFarm (ex. agent or master). See the below table to get familiar with the various sub-projects as they will be referenced later on:

Table 1.1: **Sub-Projects of PyFarm**

Sub-Project	Type	Description
pyfarm-docs	Supporting	• source code to documentation in restructured text form • **pushes to this repository will:** – build docs at https://readthedocs.org/builds/pyfarm
pyfarm-core	Operations	• common library used by all other operational type sub-projects containing standard library backports, enums, basic file handling, base logger, general utilities, and system information collection • setup.py **does not** specify all dependencies, the consuming sub-project will do this instead (ex. pyfarm-agent specifies the dependencies that the system information library needs) • **pushes to this repository will:** – build at https://travis-ci.org/pyfarm/pyfarm-core – collect coverage at https://coveralls.io/r/pyfarm/pyfarm-core
pyfarm-agent	Operations	• contains the code which runs on a remote host which can run jobs, update the master on a host's resources, and track progress of individual commands • **pushes to this repository will:** – build at https://travis-ci.org/pyfarm/pyfarm-agent – collect coverage at https://coveralls.io/r/pyfarm/pyfarm-agent • code which wraps around an individual command to provide handling for failures, log emissions, and other conditionals related to job execution
pyfarm-master	Operations	• contains the code which runs the web ui which serves as an administrative interface • contains the database models

Attention: This document is a draft

1.3.2 Requirements

This document covers the basic requirements for installation and operation of PyFarm. These are the requirements to run PyFarm itself regardless of the service being executed. These requirements do **not** cover the software PyFarm may be executing or the infrastructure required.

Summary

- **Python** Depending on the module being used, different versions of Python may be supported. Eventually Python 2.5 support will be dropped however this likely will not happen until Python 3.0 support is added. In any case, notice will be provided well in advance of a release if any of the below changes.

  Table 1.2: **Module Specific Python Version Support**

Module	Python Version
`pyfarm.core`	2.6-3.3+
`pyfarm.master`	2.7-3.3+
`pyfarm.agent`	2.6-2.7

- **Operation System** Linux, Mac, and Windows. Some features may be limited on disabled on certain platforms.
- **Memory** 64MB of memory, more may be required to run some components
- **Storage** 128MB of disk space

Database

PyFarm stores a large amount of the information it needs to operate in a relational database. Cross database support is provided by SQLAlchemy, for more information on supported databases see this document.

Python

There's not currently a *requirements.txt* file associated with this project because of the differences in dependencies between Python versions. PyFarm however can still be installed into a virtual environment using pip::

```
pip install -e . --egg
```

By doing this you `pip` will download and install all the necessary requirements for your specific Python version and platform.

Supported Software (Job Types)

PyFarm 1.0.0 provides several job types out of the box. Each of these software packages will have their own requirements as well so please visit the manufacturers website for more information.

- Maya
- Houdini
- Nuke

If you wish to request a new builtin type or check the integration status of one checkout their tickets on github

Attention: This document is a draft

1.3.3 Installation

These instructions cover the general installation of PyFarm's components. Please note they may vary between platforms though these differences are usually noted.

For reference, these instructions have been tested on the following platforms:

- Debian 7.1 64-bit
- Windows 7 Professional w/SP1 32-bit
- Windows XP Professional w/SP3 32-bit

Python Setup

Before installing PyFarm you must install Python. Below are the various steps required to install the base Python interpreter and associated libraries. When linked to a web page instead of a download please locate the installer package for your architecture and download it.

Windows

Windows requires a little extra work in order to setup Python. Unlike platforms such as Linux which make installing a compatible C-compiler a simple task Windows adds a few extra steps.

Required Downloads

1. 7-zip - to extract the ISO
2. Visual Studio 2008 Express w/SP1 - used to compile C extensions on the fly
3. Python 2.7 MSI Installer - interpreter to run PyFarm
4. setuptools - contains easy_install (used later on)

Installation

1. 7-zip - execute the installation package and follow the steps on the screen
2. **Visual Studio 2008 Express**
   (a) Right click on the ISO file and select 7-zip -> Extract to "VS2008ExpressWithSP1ENUX1504728". After this is complete you are free to delete the ISO image if you wish.
   (b) Run (double click) VS2008ExpressWithSP1ENUX1504728Setup.hta
   (c) When the setup begins you will be prompted with a few different additions to choose from. Select "Microsoft Visual C++ 2008 Express Edition"
   (d) Accept the EULA and continue forward.
   (e) **There should be two boxes checked for additional components to install:**
      - Microsoft Silverlight Runtime

- Microsoft SQL Server 2008 Express Edition

These components are not required and you are welcome to uncheck them to save time and space.

(f) Click next and continue with the installation. If the installation fails check to make sure you don't have any pending updates for windows or a reboot scheduled because of a new driver.

3. **Python**

(a) Start the installation

(b) Install for all users in the default location

(c) Open a run dialog or hold down the windows key and 'r'. When the dialog opens type 'run' (no quotes) and hit enter.

(d) When the terminal opens type 'python' and hit enter (again, no quotes). If you get something like this:

```
'python' is not recognized as an internal or external command,
operable program or batch file.
```

Then we'll need to add some things to %PATH% to continue:

Warning: be careful editing these settings, your installation

i. Right click on Computer and select Properties

ii. For Windows along the left side select you'll select "Advanced System Settings". If you're running Windows XP, skip this step.

iii. In the dialog that opens select Advanced (if it's not already) then click the "Environment Variables" button in the bottom right

iv. In the lower half of the window there's a "System Variables" section, select "Path" and then click "Edit"

v. in the "Variable value" field add this to the end:

```
;C:\Python27;C:\Python27\Scripts
```

(e) Right click on the setuptools gzipped tar (.tar.gz) and select 7-zip -> extract here

(f) Navigate down into the 'dist' directory it produces and do the same thing on the file inside that directory

(g) Once that's complete open up a command window using run and run the setup.py file. It should look something like this:

```
C:\Users\dev>python C:\Users\dev\Downloads\dist\setuptools-1.1.5\setup.py install
```

(h) Now easy_install pip:

```
C:\Users\dev>easy_install pip
Searching for pip
Reading https://pypi.python.org/simple/pip/
Best match: pip 1.4.1
Downloading https://pypi.python.org/packages/source/p/pip/pip-1.4.1.tar.gz#md5=6afbb46ael
Processing pip-1.4.1.tar.gz
Writing c:\users\dev\appdata\local\temp\easy_install-g3mjsb\pip-1.4.1\setup.cfg
Running pip-1.4.1\setup.py -q bdist_egg --dist-dir c:\users\dev\appdata\local\temp\easy_
warning: no files found matching '*.html' under directory 'docs'
```

```
warning: no previously-included files matching '*.rst' found under directory 'docs\_buil
no previously-included directories found matching 'docs\_build\_sources'
Adding pip 1.4.1 to easy-install.pth file
Installing pip-script.py script to C:\Python27\Scripts
Installing pip.exe script to C:\Python27\Scripts
Installing pip.exe.manifest script to C:\Python27\Scripts
Installing pip-2.7-script.py script to C:\Python27\Scripts
Installing pip-2.7.exe script to C:\Python27\Scripts
Installing pip-2.7.exe.manifest script to C:\Python27\Scripts

Installed c:\python27\lib\site-packages\pip-1.4.1-py2.7.egg
Processing dependencies for pip
Finished processing dependencies for pip
```

(i) Then pip install virtualenv:

```
C:\Users\dev>pip install virtualenv
Downloading/unpacking virtualenv
Downloading virtualenv-1.10.1.tar.gz (1.3MB): 1.3MB downloaded
Running setup.py egg_info for package virtualenv

    warning: no files found matching '*.egg' under directory 'virtualenv_support'
    warning: no previously-included files matching '*' found under directory 'docs\_temp
    warning: no previously-included files matching '*' found under directory 'docs\_buil
Installing collected packages: virtualenv
Running setup.py install for virtualenv

    warning: no files found matching '*.egg' under directory 'virtualenv_support'
    warning: no previously-included files matching '*' found under directory 'docs\_temp
    warning: no previously-included files matching '*' found under directory 'docs\_buil
    Installing virtualenv-script.py script to C:\Python27\Scripts
    Installing virtualenv.exe script to C:\Python27\Scripts
    Installing virtualenv.exe.manifest script to C:\Python27\Scripts
    Installing virtualenv-2.7-script.py script to C:\Python27\Scripts
    Installing virtualenv-2.7.exe script to C:\Python27\Scripts
    Installing virtualenv-2.7.exe.manifest script to C:\Python27\Scripts
Successfully installed virtualenv
Cleaning up...
```

(j) And now a quick test of the whole system. Your results will vary but it should look something like this and say "Successfully installed psutil" towards the end:

```
C:\Users\dev>virtualenv test
New python executable in test\Scripts\python.exe
Installing Setuptools.....................................................................
..............done.
Installing Pip............................................................................
.........................................................................................

C:\Users\dev>test\Scripts\activate
(test) C:\Users\dev>
(test) C:\Users\dev>pip install psutil
Downloading/unpacking psutil
You are installing an externally hosted file. Future versions of pip will default to dis
You are installing a potentially insecure and unverifiable file. Future versions of pip
Downloading psutil-1.0.1.tar.gz (159kB): 159kB downloaded
Running setup.py egg_info for package psutil
```

```
            Installing collected packages: psutil
            Running setup.py install for psutil
                building '_psutil_mswindows' extension
                C:\Program Files\Microsoft Visual Studio 9.0\VC\BIN\cl.exe /c /nologo /Ox /MD /W3 /G
            se\psutil/_psutil_mswindows.obj
                _psutil_mswindows.c
                psutil/_psutil_mswindows.c(307) : warning C4013: 'get_process_info' undefined; assum
                psutil/_psutil_mswindows.c(568) : warning C4047: 'function' : 'LPSTR' differs in lev
                psutil/_psutil_mswindows.c(568) : warning C4024: 'GetProcessImageFileNameA' : differ
                psutil/_psutil_mswindows.c(602) : warning C4133: 'function' : incompatible types - f
                psutil/_psutil_mswindows.c(2091) : warning C4047: 'function' : 'PDWORD_PTR' differs
                psutil/_psutil_mswindows.c(2091) : warning C4024: 'GetProcessAffinityMask' : differ
                psutil/_psutil_mswindows.c(2091) : warning C4047: 'function' : 'PDWORD_PTR' differs
                psutil/_psutil_mswindows.c(2091) : warning C4024: 'GetProcessAffinityMask' : differ
                psutil/_psutil_mswindows.c(2413) : warning C4005: '_ARRAYSIZE' : macro redefinition
                        C:\Program Files\Microsoft SDKs\Windows\v6.0A\include\winnt.h(1021) : see pr
                psutil/_psutil_mswindows.c(2482) : warning C4047: 'function' : 'LPSTR' differs in le
                psutil/_psutil_mswindows.c(2482) : warning C4024: 'GetVolumeInformationA' : differen
                C:\Program Files\Microsoft Visual Studio 9.0\VC\BIN\cl.exe /c /nologo /Ox /MD /W3 /G
            psutil/_psutil_common.obj
                _psutil_common.c
                C:\Program Files\Microsoft Visual Studio 9.0\VC\BIN\cl.exe /c /nologo /Ox /MD /W3 /G
            -2.7\Release\psutil/arch/mswindows/process_info.obj
                process_info.c
                psutil/arch/mswindows/process_info.c(36) : warning C4013: 'AccessDenied' undefined;
                psutil/arch/mswindows/process_info.c(36) : warning C4047: 'return' : 'HANDLE' differ
                psutil/arch/mswindows/process_info.c(42) : warning C4013: 'NoSuchProcess' undefined;
                C:\Program Files\Microsoft Visual Studio 9.0\VC\BIN\cl.exe /c /nologo /Ox /MD /W3 /G
            n32-2.7\Release\psutil/arch/mswindows/process_handles.obj
                process_handles.c
                psutil/arch/mswindows/process_handles.c(203) : warning C4022: 'NtDuplicateObject' :
                C:\Program Files\Microsoft Visual Studio 9.0\VC\BIN\cl.exe /c /nologo /Ox /MD /W3 /G
            \Release\psutil/arch/mswindows/security.obj
                security.c
                psutil/arch/mswindows/security.c(86) : warning C4996: 'strcpy': This function or var
                C:\Program Files\Microsoft Visual Studio 9.0\VC\BIN\link.exe /DLL /nologo /INCREMENT
            hlpapi.lib wtsapi32.lib /EXPORT:init_psutil_mswindows build\temp.win32-2.7\Release\psuti
            2-2.7\Release\psutil/arch/mswindows/process_handles.obj build\temp.win32-2.7\Release\psu
            ld\temp.win32-2.7\Release\psutil\_psutil_mswindows.pyd.manifest
                Creating library build\temp.win32-2.7\Release\psutil\_psutil_mswindows.lib and objec

            Successfully installed psutil
            Cleaning up...

            (test) C:\Users\dev>python -c "import psutil; print psutil"
            <module 'psutil' from 'C:\Users\dev\test\lib\site-packages\psutil\__init__.pyc'>
```

Linux

Linux generally won't require very much to be done to get started and in most cases is already setup for you. That said, it wouldn't hurt to run the steps below to be sure. The following commands will need to be executed either as root or using the sudo command.

Debian Based Systems

```
aptitude -y install python python-setuptools python-virtualenv build-essential
```

Red Hat Based Systems **TODO** instructions needed here

1.3.4 License

PyFarm is licensed under the terms and conditions of the Apache 2.0 license. The original text of this license may either be downloaded or viewed on the web. A formatted version with the same text as the original is also available below.

1. Definitions

License shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

Licensor shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

Legal Entity shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

You (or **Your**) shall mean an individual or Legal Entity exercising permissions granted by this License.

Source form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

Object form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

Work shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

Derivative Works shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

Contribution shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

Contributor shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly

display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License

Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution

You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or
    Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices
    stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works
    that You distribute, all copyright, patent, trademark, and
    attribution notices from the Source form of the Work,
    excluding those notices that do not pertain to any part of
    the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its
    distribution, then any Derivative Works that You distribute must
    include a readable copy of the attribution notices contained
    within such NOTICE file, excluding those notices that do not
    pertain to any part of the Derivative Works, in at least one
    of the following places: within a NOTICE text file distributed
    as part of the Derivative Works; within the Source form or
    documentation, if provided along with the Derivative Works; or,
    within a display generated by the Derivative Works, if and
    wherever such third-party notices normally appear. The contents
    of the NOTICE file are for informational purposes only and
    do not modify the License. You may add Your own attribution
    notices within Derivative Works that You distribute, alongside
    or as an addendum to the NOTICE text from the Work, provided
    that such additional attribution notices cannot be construed
    as modifying the License.

5. Submission of Contributions

Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks

This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty

Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability

In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability

While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

CHAPTER 2

Indices and tables

- genindex
- search

/api

`POST /api/v1/foo/<str:item> HTTP/1.1`, 5

P