

---

# **pyfarm.master Documentation**

*Release 0.8.7*

**Oliver Palmer, Guido Winkelmann**

September 25, 2015



<b>1</b>	<b>Commands</b>	<b>3</b>
1.1	Standard Commands . . . . .	3
1.2	Development Commands . . . . .	3
<b>2</b>	<b>Configuration Files</b>	<b>5</b>
2.1	Master . . . . .	5
2.2	Models . . . . .	9
2.3	Scheduler . . . . .	13
<b>3</b>	<b>pyfarm.master package</b>	<b>19</b>
3.1	Subpackages . . . . .	19
3.2	Submodules . . . . .	78
3.3	Module contents . . . . .	84
<b>4</b>	<b>pyfarm.models package</b>	<b>85</b>
4.1	Subpackages . . . . .	85
4.2	Submodules . . . . .	92
4.3	Module contents . . . . .	107
<b>5</b>	<b>pyfarm.scheduler package</b>	<b>109</b>
5.1	Submodules . . . . .	109
5.2	Module contents . . . . .	109
<b>6</b>	<b>Indices and tables</b>	<b>111</b>
	<b>HTTP Routing Table</b>	<b>113</b>
	<b>Python Module Index</b>	<b>115</b>



This package contains the models, web interface, APIs, and backend components necessary to scheduler and allocate jobs on PyFarm.

**Contents**



---

## Commands

---

### 1.1 Standard Commands

#### 1.1.1 pyfarm-create-tables

```
usage: pyfarm-tables [-h] [--echo] [--drop-all] [--no-create-tables]

Creates PyFarm's tables

optional arguments:
  -h, --help            show this help message and exit
  --echo                If provided then echo the SQL queries being made
  --drop-all           If provided all tables will be dropped from the database
                       before they are created.
  --no-create-tables   If provided then no tables will be created.
```

### 1.2 Development Commands

#### 1.2.1 pyfarm-master

```
usage: pyfarm-master [-h] [--drop-all] [--create-all]
                   [--confirm-drop CONFIRM_DROP]
                   [--allow-agent-loopback-addresses]

optional arguments:
  -h, --help            show this help message and exit
  --drop-all, -D       drop the existing tables before starting
  --create-all, -C     create all tables before starting
  --confirm-drop CONFIRM_DROP
  --allow-agent-loopback-addresses
```





---

## Configuration Files

---

Below are the configuration files for this subproject. These files are installed along side the source code when the package is installed. These are only the defaults however, you can always override these values in your own environment. See the [Configuration](#) object documentation for more detailed information.

### 2.1 Master

The below is the current configuration file for the agent. This file lives at `pyfarm/master/etc/master.yml` in the source tree.

```
1 # Configures if the underlying Flask application and other libraries
2 # should run in debug mode.
3 #
4 # **Production Note**: This value should always be false.
5 debug: true
6
7
8 # Enables verbose output of loggers associated with the
9 # job queue.
10 debug_queue: false
11
12
13 # The URL to access the database. For debugging and development
14 # a local SQLite database is used but for production other databases,
15 # such as MySQL or PostgreSQL which PyFarm runs tests against, should
16 # be used.
17 #
18 # For more information on the expected format of this variable
19 # see SQLAlchemy's documentation:
20 # https://sqlalchemy.readthedocs.org/en/latest/core/engines.html#database-urls
21 database: "sqlite:///pyfarm.sqlite"
22
23
24 # Where to store runtime statistics. Same format as "database"
25 statistics_database: "sqlite:///pyfarm-statistics.sqlite"
26
27
28 # The broker that PyFarm's scheduler should use. For debugging and
29 # development running Redis is the simplest. For large deployments, or
30 # to understand the format of this variable, see:
31 # http://celery.readthedocs.org/en/latest/configuration.html#broker-url
```

```
32 scheduler_broker: "redis://"
33
34
35 # The URL the master is running on. This is used to form internal
36 # urls and other information.
37 base_url: http://127.0.0.1:5000/
38
39
40 # The name of this render farm, normally this can remain the default value. If
41 # you're running multiple farms this allows you to only accept agents to your
42 # master that match your current farm name.
43 farm_name: ""
44
45
46 # The secret key which is used by several components of Flask
47 # for form validation, salting of secrets, etc.
48 #
49 # Production Note: This value should be random, consistent between
50 # frontends, and kept secret. Do not use the value below for
51 # production.
52 secret_key: pyfarm
53
54
55 # The key used for signing the csrf token.
56 #
57 # Production Note: This value should be random, consistent between
58 # frontends, and kept secret. Do not use the value below for
59 # production.
60 csrf_session_key: pyfarm
61
62
63 # The prefix of the URL from which the API will operate on. This should
64 # not generally be changed unless you are operating different versions
65 # of the API at the same time from one web server.
66 api_prefix: /api/v1
67
68
69 # The URL template we use to communicate with the agent.
70 agent_api_url_template: http://{host}:{port}/api/v1
71
72
73 # Enables or disable the login functionality. This can be used when
74 # debugging or doing development but should not be changed for
75 # production.
76 login_disabled: false
77
78
79 # The amount of time the 'remember me' cookie should persist. The keys
80 # and values here are passed into a `timedelta` object as keywords.
81 cookie_duration:
82     hours: 16
83
84
85 # When true json output from the APIs will be reformatted to
86 # be more human readable.
87 pretty_json: false
88
89
```

```
90 # When true all SQLAlchemy queries will be echoed. This is useful
91 # for debugging the SQL statements being run and to get an idea of
92 # what the underlying ORM may be doing.
93 echo_sql: false
94
95
96 # When true the application will automatically create users in
97 # PyFarm's database if they do not exist already. Setting this
98 # to false will cause an exception to be raised if the user in
99 # question does not exist.
100 autocreate_users: true
101
102
103 # When autocreating users, use this email address as a template. For example:
104 #     "{username}@example.com"
105 # Not setting this value and setting `autocreate_users` to true will result
106 # in a user's email address not being set on a newly created user.
107 autocreate_user_email: null
108
109
110 # When provided an integer this many seconds will elapse after a job
111 # has completed before it is deleted.
112 default_job_delete_time: null
113
114
115 # The format for timestamps in the user interface.
116 timestamp_format: "YYYY-MM-DD HH:mm:ss"
117
118
119 # The directory to store updates for agents. This will use `temp` above
120 # as the base directory.
121 agent_updates_dir: ${temp}/pyfarm-updates
122
123
124 # Optional directory to serve GET requests for agent updates
125 # from. This is different from `agent_updates_dir` in that it's
126 # only used when an agent is requested a file to update from. This
127 # can be useful when you're caching requests or doing something with
128 # the update files prior to them being requested by the agent.
129 agent_updates_webdir: null
130
131
132 # The directory to store downloaded logs in.
133 #
134 # Production Note: For production it's probably best if these are kept
135 # in a persistent location rather than $temp.
136 tasklogs_dir: ${temp}/task_logs
137
138
139 # The address the Flask application should listen on. This is only important
140 # when running the application in a standalone mode of operation. By default
141 # this will only listen locally but could be changed to listen on
142 # a specific adapter or `0.0.0.0` for all addresses.
143 flask_listen_address: 127.0.0.1
144
145
146 # When true all database tables will be dropped prior to setting
147 # up the application. This is useful for development purposes only
```

```
148 # and should not be used in production. There's also the `pyfarm-tables`
149 # command line tool which can be used to create or drop tables.
150 dev_db_drop_all: false
151
152
153 # When true we'll attempt to create any missing database tables
154 # prior to the application starting. This is useful for development
155 # purposes only and should not be used in production. There's also
156 # the `pyfarm-tables` command line tool which can be used to create
157 # or drop tables.
158 dev_db_create_all: false
159
160
161 # When true the application will be instanced as 'app' in the
162 # pyfarm.master.entrypoints module. When running behind something
163 # like uwsgi this should be true.
164 instance_application: false
165
166 ##
167 ## BEGIN Queue defaults
168 ##
169
170
171 # The default priority for a newly created job queue.
172 queue_default_priority: 0
173
174
175 # The default weight of a newly created job queue.
176 queue_default_weight: 10
177
178
179 # The minimum and maximum priority any queue can have. This is
180 # used by the models for validation purposes.
181 queue_min_priority: -1000
182 queue_max_priority: 1000
183
184 ##
185 ## END Queue defaults
186 ##
187
188 ##
189 ## BEGIN Job Type defaults
190 ##
191
192 # The maximum number of tasks for the given job type
193 # to send to an agent at once.
194 job_type_max_batch: 1
195
196
197 # When batching and this value is true frames will be batched
198 # in contiguous groups.
199 job_type_batch_contiguous: true
200
201 ##
202 ## END Job Type defaults
203 ##
```

## 2.2 Models

The below is the current configuration file for job types. This file lives at `pyfarm/models/etc/models.yml` in the source tree.

```

1  ##
2  ## BEGIN Database Table Names
3  ##
4
5  # Prefix used in the construction of all table names. See the variables
6  # below for uage.
7  table_prefix: ""
8
9
10 # The name of the table for software items
11 table_software: ${table_prefix}software
12
13 # The name of the table for software versions
14 table_software_version: ${table_software}_versions
15
16 # The name of the table used for tagging
17 table_tag: ${table_prefix}tags
18
19 # The name of the table storing agent entries
20 table_agent: ${table_prefix}agents
21
22 # The name of the table which associates agents and software versions
23 table_agent_software_version_assoc: ${table_prefix}agent_software_version_associations
24
25 # The name of the table which associates agents and tags
26 table_agent_tag_assoc: ${table_prefix}agent_tag_associations
27
28 # The name of the table which associated agents and mac addresses
29 table_agent_mac_address: ${table_prefix}agent_mac_addresses
30
31 # The name of the table containing jobs
32 table_job: ${table_prefix}jobs
33
34 # The name of the table containing job types
35 table_job_type: ${table_prefix}jobtypes
36
37 # The name of the table containing job type versions
38 table_job_type_version: ${table_prefix}jobtype_versions
39
40 # The name of the table which associates jobs and tags.
41 table_job_tag_assoc: ${table_prefix}job_tag_associations
42
43 # The name of the table which associates job and tag requirements
44 table_job_tag_req: ${table_prefix}job_tag_requirements
45
46 # The name of the table which associates inter-job dependencies
47 table_job_dependency: ${table_prefix}job_dependencies
48
49 # The name of the table which associates job and software requirements
50 table_job_software_req: ${table_prefix}job_software_requirements
51
52 # The name of the table containing information about users to be notified
53 # of status changes form jobs

```

```
54 table_job_notified_users: ${table_prefix}notified_users
55
56 # The name of the table which associates software requirements and jobs
57 table_job_type_software_req: ${table_prefix}jobtype_software_requirements
58
59 # The name of the table containing tasks
60 table_task: ${table_prefix}tasks
61
62 # The name of the table containing user information
63 table_user: ${table_prefix}users
64
65 # The name of the table containing role information
66 table_role: ${table_prefix}roles
67
68 # The name of the table which associates users and roles
69 table_user_role: ${table_prefix}user_roles
70
71 # The name of the table containing the job queues
72 table_job_queue: ${table_prefix}job_queues
73
74 # The name of the table containing job groups
75 table_job_group: ${table_prefix}job_groups
76
77 # The name of the table containing path mappings
78 table_path_map: ${table_prefix}path_maps
79
80 # The name of the table containing task logs
81 table_task_log: ${table_prefix}task_logs
82
83 # The name of the table containing associations between task
84 # logs and jobs
85 table_task_log_assoc: ${table_prefix}task_log_associations
86
87 # The name of the table containing GPU information for agents
88 table_gpu: ${table_prefix}gpus
89
90 # The name of the table containing associations between agents and GPUs
91 table_gpu_in_agent: ${table_prefix}gpu_agent_associations
92
93 # The name of the table storing which tasks where failed on an agent
94 table_failed_task_in_agent: ${table_prefix}failed_tasks_in_agents
95
96 # The name of the table containing the disks of the agents
97 table_agent_disk: ${table_prefix}agent_disks
98
99 table_statistics_agent_count: ${table_prefix}agent_counts
100
101 table_statistics_task_event_count: ${table_prefix}task_event_counts
102
103 table_statistics_task_count: ${table_prefix}task_counts
104
105 ##
106 ## END Database Table Names
107 ##
108
109 ##
110 ## BEGIN Database Model Constraints
111 ##
```

```
112
113 # There's some validation that happens when an agent is added to the
114 # database. One of the checks we have is to ensure the agent's address
115 # is a remote address which a loopback address normally is not considered
116 # 'remote'. Changing this value to true disable this and will allow
117 # agents from a local address to connect.
118 allow_agents_from_loopback: false
119
120
121 # The maximum length of a tag
122 max_tag_length: 64
123
124
125 # The maximum length of a hostname
126 max_hostname_length: 255
127
128
129 # The maximum length of a job group's name.
130 max_jobgroup_name_length: 255
131
132
133 # The maximum length of the operating system's name for an agent.
134 max_osname_length: 128
135
136
137 # The maximum length of an agent's CPU name
138 max_cpuname_length: 128
139
140
141 # Not Implemented The default amount of ram the agent is allowed to
142 # allocate towards work. A value of 1.0 would allow the agent to be
143 # assigned as much work as the system's ram would allow.
144 agent_ram_allocation: .8
145
146
147 # Not Implemented Based on load, this is the default amount of CPU space
148 # an agent is allowed to occupy with work.
149 agent_cpu_allocation: 1.0
150
151
152 # The minimum and maximum ports an agent can connect from
153 agent_min_port: 1024
154 agent_max_port: 65535
155
156
157 # The minimum and maximum CPUs an agent can declare
158 # These values also drive the min/max number of CPUS job is allowed to request.
159 agent_min_cpus: 1
160 agent_max_cpus: 256
161
162
163 # The minimum and maximum amount of RAM, in megabytes, an agent can declare.
164 # These values also drive the min/max amount of ram a job is allowed to request.
165 agent_min_ram: 16
166 agent_max_ram: 262144
167
168
169 # The default weight given to a job for use in the queue.
```

```
170 queue_default_weight: 10
171
172
173 # The maximum length a job's title is allowed to be
174 jobtitle_max_length: 255
175
176
177 # The global default batch size for all new jobs.
178 job_default_batch: 1
179
180
181 # The global default number of times a job will requeue
182 # for tailed tasks. 0 will never requeue, -1 will
183 # requeue indefinitely.
184 job_requeue_default: 3
185
186
187 # The global default minimum number of CPUs a job may execute
188 # on. 0 will disable the minimum, -1 will force an entire agent
189 # to be exclusive to a job's task.
190 job_default_cpus: 1
191
192
193 # The global default amount of ram that's required to be free on
194 # host in order for a task of a job to run on a given agent. A
195 # value of 0 will not require a minimum, -1 will force the agent's
196 # entire ram to be allocated to the given task.
197 job_default_ram: 32
198
199
200 # The maximum length a path mapping is allowed to be.
201 max_path_length: 512
202
203
204 # The maximum length a GPU name is allowed to be.
205 max_gpu_name_length: 128
206
207
208 # The maximum length a queue name is allowed to be.
209 max_queue_name_length: 255
210
211
212 # The maximum length of a queue's path
213 max_queue_path_length: 1024
214
215
216 # The maximum length of a job type's name
217 job_type_max_name_length: 64
218
219
220 # The maximum length of a job type's class name
221 job_type_max_class_name_length: 64
222
223
224 # The maximum length of a username
225 max_username_length: 255
226
227
```



```

228 # The maximum length of an email address
229 max_email_length: 255
230
231
232 # The maximum length of a role name
233 max_role_length: 128
234
235 # The maximum length of a mountpoint for agent disks
236 max_mountpoint_length: 255
237
238 # The maximum length of the function name to discover the presence of a
239 # software version on an agent
240 max_discovery_function_name_length: 255
241
242 ##
243 ## END Database Model Constraints
244 ##

```

## 2.3 Scheduler

The below is the current configuration file for job types. This file lives at `pyfarm/scheduler/etc/scheduler.yml` in the source tree.

```

1 ##
2 ## BEGIN Scheduler Settings
3 ##
4
5 # The user agent the scheduler will use when connecting to
6 # an agent. Do not change this value unless the agent is
7 # updated to reflect the change made here.
8 master_user_agent: "PyFarm/1.0 (master)"
9
10
11 # How often the scheduler should run and poll agents. The keys and
12 # values here are passed into a `timedelta` object as keywords.
13 agent_poll_interval:
14     seconds: 30
15
16
17 # How often the scheduler should run and and assign tasks. The keys and
18 # values here are passed into a `timedelta` object as keywords.
19 assign_tasks_interval:
20     minutes: 4
21
22
23 # How often orphaned task logs should be cleaned up on disk. The keys and
24 # values here are passed into a `timedelta` object as keywords.
25 orphaned_log_cleanup_interval:
26     hours: 1
27
28
29 # How often we should attempt to compress old task logs. The keys and
30 # values here are passed into a `timedelta` object as keywords.
31 compress_log_interval:
32     minutes: 10
33

```

```
34
35 # How often old jobs should be deleted. Please note this only marks
36 # jobs as to be deleted and does not actually perform the deletion
37 # itself. See the ``delete_job_interval`` setting which will actually
38 # trigger the deletion of jobs. The keys and values here are passed
39 # into a `timedelta` object as keywords.
40 autodelete_old_job_interval:
41     hours: 1
42
43
44 # How often the scheduler which deletes jobs should run. The keys and values
45 # here are passed into a `timedelta` object as keywords.
46 delete_job_interval:
47     minutes: 5
48
49
50 # Used when polling agents to determine if we should or should not
51 # reach out to an agent. This is used in combination with the agent's
52 # `last_heard_from` column, it's state and number of running tasks. The keys
53 # and values here are passed into a `timedelta` object as keywords.
54 poll_busy_agents_interval:
55     minutes: 5
56
57
58 # Used when polling agents to determine if we should or should not
59 # reach out to an agent. This is used in combination with an agent's
60 # `last_head_from` column, state and running task count. The keys
61 # and values here are passed into a `timedelta` object as keywords.
62 poll_idle_agents_interval:
63     hours: 1
64
65
66 # Used when polling agents to determine if an agent is considered
67 # offline or not after a given period of time without communication. The keys
68 # and values here are passed into a `timedelta` object as keywords.
69 poll_offline_agents_interval:
70     hours: 2
71
72
73 # A directory where lock files for the scheduler can be found.
74 scheduler_lockfile_base: ${temp}/scheduler_lock
75
76 # The number of times an SQL transaction error should be retried.
77 transaction_retries: 10
78
79 # The number of seconds we wait for a request to an agent to respond. An
80 # exception is raised if we exceed this amount.
81 agent_request_timeout: 10
82
83 # When true the queue will prefer to assign work
84 # for jobs which are already running.
85 queue_prefer_running_jobs: true
86
87 # Whether to use an agent's total RAM instead of reported free RAM to determine
88 # whether or not it can run a task.
89 use_total_ram_for_scheduling: false
90
91 ##
```

```
92  ## END Scheduler Settings
93  ##
94
95  ##
96  ## BEGIN Email Server Settings
97  ##
98
99  # The smtp server used to send email notifications. Note that setting
100 # this value to null or leaving it blank will disable email notifications.
101 smtp_server: localhost
102
103
104 # Port to connect to the smtp server on. The default port, 0, will
105 # cause the underlying library to use the default smtp port.
106 smtp_port: 0
107
108
109 # Optional login credentials for the smtp server. The default value
110 # [null, null] means no username and password is required.
111 smtp_login: [null, null]
112
113
114 # The default address from which all emails from the scheduler will
115 # originate.
116 from_email: pyfarm@localhost
117
118 ##
119 ## END Email Server Settings
120 ##
121
122 ##
123 ## BEGIN Email Template Settings
124 ##
125
126 # General note about the settings below. The brackets, {{ }}
127 # are used by the templating system for string substitution. For example,
128 # {{ job.title }} would substitute in the string found on the `title` column
129 # of a job model. For more information on template formatting, see Jinja's
130 # documentation: http://jinja.pocoo.org/docs/
131 # Finally, for multi-line strings follow this syntax:
132 #   foobar:
133 #     |
134 #     This is a multi-line
135 #     string. It's indentation
136 #
137 #     and whitespace will be preserved.
138
139
140 # The template email subject line used for a succesful job.
141 success_subject: Job {{ job.title }} completed successfully
142
143
144 # The template body of an email for a succesful job
145 success_body:
146 |
147     {{ job.jobtype_version.jobtype.name }} job {{ job.title }} (id {{ job.id }})
148     has completed successfully on {{ job.time_finished.isoformat() }}.
149
```

```
150 Job: {{ job.url }}
151
152 {% if job.output_link %}
153 Output: {{ job.output_link }}
154 {% endif %}
155
156 Sincerely,
157     The PyFarm render manager
158
159
160 # The template email subject line used for a failed job.
161 failed_subject: Job {{ job.title }} failed
162
163
164 # The template email body for a failed job.
165 failed_body:
166 |
167     {{ job.jobtype_version.jobtype.name }} job {{ job.title }}
168     (id {{ job.id }}) has failed on
169     {{ job.time_finished.isoformat() }}.
170
171 Job: {{ job.url }}
172
173 {% if job.output_link %}
174 Output:
175
176     {{ job.output_link }}
177 {% endif %}
178
179 {% if failed_log_urls %}
180 Log(s) for failed tasks:
181 {% for url in failed_log_urls %}
182     {{url}}
183 {% endfor%}
184 {% endif %}
185
186 Sincerely,
187     The PyFarm render manager
188
189
190 # The template email subject line used for a deleted job. Supported
191 # template values are:
192 #     {job_title} - The title of the job being deleted
193 deleted_subject: Job {job_title} deleted
194
195 # The template email body for a deleted job. Supported template
196 # values are:
197 #     {job_title} - The title of the job deleted
198 #     {job_id} - The id of the job deleted
199 #     {jobtype_name} - The name of the job type used
200 deleted_body:
201 |
202     {jobtype_name} job {job_title} has been deleted.
203
204 Sincerely,
205     The PyFarm render manager
206
207
```

```
208 ##
209 ## END Email Template Settings
210 ##
211
212
213 ##
214 ## BEGIN Statistics Gathering Settings
215 ##
216
217 # Whether or not to gather data for runtime statistics
218 enable_statistics: true
219
220
221 agent_count_interval:
222     hours: 1
223
224 task_event_count_consolidate_interval:
225     minutes: 15
226
227 task_count_interval:
228     minutes: 15
229
230 ##
231 ## END Statistics Gathering Settings
232 ##
```



---

## pyfarm.master package

---

### 3.1 Subpackages

#### 3.1.1 pyfarm.master.api package

##### Submodules

##### pyfarm.master.api.agent\_updates module

**Agent Updates** The API allows access to agent update packages, possibly through redirects

**class** `pyfarm.master.api.agent_updates.AgentUpdatesAPI`

Bases: `flask.views.MethodView`

**get** (*version*)

A GET to this endpoint will return the update package as a zip file the specified version

**GET** `/api/v1/agents/updates/<string:version>` **HTTP/1.1**

##### Request

```
PUT /api/v1/agents/updates/1.2.3 HTTP/1.1
Accept: application/zip
```

##### Response

```
HTTP/1.1 200 OK
Content-Type: application/zip

<binary data>
```

**Statuscode 200** The update file was found and is returned

**Statuscode 301** The update can be found under a different URL

**Statuscode 400** there was something wrong with the request (such as an invalid version number specified or the mime type not being application/zip)

**methods** = ['GET', 'PUT']

**put** (*version*)

A PUT to this endpoint will upload a new version of pyfarm-agent to be used for agent auto-updates. The update must be a zip file.

**PUT /api/v1/agents/updates/<string:version> HTTP/1.1**  
**Request**

```
PUT /api/v1/agents/updates/1.2.3 HTTP/1.1
Content-Type: application/zip

<binary data>
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json
```

**Statuscode 201** The update was put in place

**Statuscode 400** there was something wrong with the request (such as an invalid version number specified or the mime type not being application/zip)

### pyfarm.master.api.agents module

**Agents** Contained within this module are an API handling functions which can manage or query agents using JSON.

**class** pyfarm.master.api.agents.**AgentIndexAPI**

Bases: flask.views.MethodView

**get** ()

A GET to this endpoint will return a list of known agents, with id and name.

**GET /api/v1/agents/ HTTP/1.1**

**Request**

```
GET /api/v1/agents/ HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "hostname": "agent1",
    "id": "dd0c6da2-0c91-42cf-a82f-6d503aae43d3"
  },
  {
    "hostname": "agent2",
    "id": "8326779e-90b5-447c-8da8-1eaa154771d9"
  },
  {
    "hostname": "agent3.local",
    "id": "14b28230-64a1-4b62-803e-5fd1baa209e4"
  }
]
```

**Request (with filters)**



```
GET /api/v1/agents/?min_ram=4096&min_cpus=4 HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json
[
  {
    "hostname": "foobar",
    "port": 50000,
    "remote_ip": "127.0.0.1",
    "id": "e20bae92-6472-442e-98a8-0ea4c9ee41cd"
  }
]
```

**Qparam min\_ram** If set, list only agents with min\_ram ram or more

**Qparam max\_ram** If set, list only agents with max\_ram ram or less

**Qparam min\_cpus** If set, list only agents with min\_cpus cpus or more

**Qparam max\_cpus** If set, list only agents with max\_cpus cpus or less

**Qparam hostname** If set, list only agents matching hostname

**Qparam remote\_ip** If set, list only agents matching remote\_ip

**Qparam port** If set, list only agents matching port.

**Statuscode 200** no error, host may or may not have been found

**methods** = ['GET', 'POST']

**post** ()

A POST to this endpoint will either create or update an existing agent. The port and id columns will determine if an agent already exists.

- If an agent is found matching the port and id columns from the request the existing model will be updated and the resulting data and the OK code will be returned.
- If we don't find an agent matching the port and id however a new agent will be created and the resulting data and the CREATED code will be returned.

**Note:** The remote\_ip field is not required and should typically not be included in a request. When not provided remote\_ip is populated by the server based off of the ip of the incoming request. Providing remote\_ip in your request however will override this behavior.

**POST /api/v1/agents/ HTTP/1.1**

**Request**

```
POST /api/v1/agents/ HTTP/1.1
Accept: application/json

{
  "cpu_allocation": 1.0,
  "cpus": 14,
  "free_ram": 133,
  "hostname": "agent1",
```

```
"id": "6a0c11df-660f-4c1e-9fb4-5fe2b8cd2437",
"remote_ip": "10.196.200.115",
"port": 64994,
"ram": 2157,
"ram_allocation": 0.8,
"state": 8
}
```

#### Response (agent created)

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

{
  "cpu_allocation": 1.0,
  "cpus": 14,
  "use_address": "remote",
  "free_ram": 133,
  "time_offset": 0,
  "hostname": "agent1",
  "id": "6a0c11df-660f-4c1e-9fb4-5fe2b8cd2437",
  "port": 64994,
  "ram": 2157,
  "ram_allocation": 0.8,
  "state": "online",
  "remote_ip": "10.196.200.115"
}
```

#### Response (existing agent updated)

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "cpu_allocation": 1.0,
  "cpus": 14,
  "use_address": "remote",
  "free_ram": 133,
  "time_offset": 0,
  "hostname": "agent1",
  "id": "6a0c11df-660f-4c1e-9fb4-5fe2b8cd2437",
  "port": 64994,
  "ram": 2157,
  "ram_allocation": 0.8,
  "state": "online",
  "remote_ip": "10.196.200.115"
}
```

**Statuscode 201** a new agent was created

**Statuscode 200** an existing agent is updated with data from the request

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

```
class pyfarm.master.api.agents.SingleAgentAPI
    Bases: flask.views.MethodView
```

API view which is used for retrieving information about and updating single agents.

**delete** (*agent\_id*)

Delete a single agent

**DELETE** /api/v1/agents/ (*uuid: agent\_id*) HTTP/1.1

**Request (agent exists)**

```
DELETE /api/v1/agents/b25ee7eb-9586-439a-b131-f5d022e0d403 HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 204 NO CONTENT
Content-Type: application/json
```

**Statuscode 204** the agent was deleted or did not exist

**get** (*agent\_id*)

Return basic information about a single agent

**GET** /api/v1/agents/ (*str: agent\_id*) HTTP/1.1

**Request (agent exists)**

```
GET /api/v1/agents/4eefca76-1127-4c17-a3df-c1a7de685541 HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "cpu_allocation": 1.0,
  "cpus": 14,
  "use_address": 311,
  "free_ram": 133,
  "time_offset": 0,
  "hostname": "agent1",
  "id": "322360ad-976f-4103-9acc-a811d43fd24d",
  "ip": "10.196.200.115",
  "port": 64994,
  "ram": 2157,
  "ram_allocation": 0.8,
  "state": 202,
  "remote_ip": "10.196.200.115"
}
```

**Request (no such agent)**

```
GET /api/v1/agents/4eefca76-1127-4c17-a3df-c1a7de685541 HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 404 NOT FOUND
Content-Type: application/json

{"error": "Agent `4eefca76-1127-4c17-a3df-c1a7de685541` not "
      "found"}
```

**Statuscode 200** no error

**Statuscode 400** something within the request is invalid

**Statuscode 404** no agent could be found using the given id

**methods** = ['DELETE', 'GET', 'POST']

**post** (*agent\_id*)

Update an agent's columns with new information by merging the provided data with the agent's current definition in the database.

**POST** /api/v1/agents/ (*str: agent\_id*) **HTTP/1.1**

**Request**

```
POST /api/v1/agents/29d466a5-34f8-408a-b613-e6c2715077a0 HTTP/1.1
Accept: application/json

{"ram": 1234}
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "cpu_allocation": 1.0,
  "cpus": 14,
  "use_address": 311,
  "free_ram": 133,
  "time_offset": 0,
  "hostname": "agent1",
  "id": "29d466a5-34f8-408a-b613-e6c2715077a0",
  "ip": "10.196.200.115",
  "port": 64994,
  "ram": 1234,
  "ram_allocation": 0.8,
  "state": "running",
  "remote_ip": "10.196.200.115"
}
```

**Statuscode 200** no error

**Statuscode 400** something within the request is invalid

**Statuscode 404** no agent could be found using the given id

**class** pyfarm.master.api.agents.**SingleSoftwareInAgentAPI**

Bases: flask.views.MethodView

**delete** (*agent\_id, software\_name, version\_name*)

A DELETE to this endpoint will remove the specified software version from the list of supported software in this agent

**DELETE** /api/v1/agents/<str:agent\_id>/software/<str:software>/versions/<str:version>  
**Request**

```
DELETE /api/v1/agents/bbf55143-f2b1-4c15-9d41-139bd8057931/software/Blender/versions/2.7
Accept: application/json
```

**Response**

```
HTTP/1.1 200 NO CONTENT
```

**Statuscode 204** the software version has been removed from the supported versions on this agent or has not been on the list in the first place

**Statuscode 404** agent not found

**methods** = ['DELETE']

**class** pyfarm.master.api.agents.**SoftwareInAgentIndexAPI**  
 Bases: flask.views.MethodView

**get** (*agent\_id*)

A GET to this endpoint will return a list of all software versions available on this agent.

**GET** /api/v1/agents/<str:agent\_id>/software/ HTTP/1.1  
**Request**

```
GET /api/v1/agents/bbf55143-f2b1-4c15-9d41-139bd8057931/software/ HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "software": "Blender",
    "version": "2.72"
  }
]
```

**Statuscode 200** no error

**Statuscode 404** agent not found

**methods** = ['GET', 'POST']

**post** (*agent\_id*)

A POST to this endpoint will mark the given version of the given software as available on this agent.

**POST** /api/v1/agents/<str:agent\_id>/software/ HTTP/1.1  
**Request**

```
POST /api/v1/agents/bbf55143-f2b1-4c15-9d41-139bd8057931/software/ HTTP/1.1
Accept: application/json

{
  "software": "Blender",
```

```
    "version": "2.72"
}
```

### Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "software": "Blender",
  "version": "2.72"
}
```

**Statuscode 200** no error

**Statuscode 400** the request contained unknown keys or required keys were missing.

**Statuscode 404** agent not found

```
class pyfarm.master.api.agents.TasksInAgentAPI
```

```
    Bases: flask.views.MethodView
```

```
    get (agent_id)
```

A GET to this endpoint will return a list of all tasks assigned to this agent.

```
    GET /api/v1/agents/<str:agent_id>/tasks/ HTTP/1.1
```

### Request

```
GET /api/v1/agents/bbf55143-f2b1-4c15-9d41-139bd8057931/tasks/ HTTP/1.1
Accept: application/json
```

### Response

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "state": "assign",
    "priority": 0,
    "job": {
      "jobtype": "TestJobType",
      "id": 1,
      "title": "Test Job",
      "jobtype_version": 1,
      "jobtype_id": 1
    },
    "hidden": false,
    "time_started": null,
    "project_id": null,
    "frame": 2.0
    "agent_id": "bbf55143-f2b1-4c15-9d41-139bd8057931",
    "id": 2,
    "attempts": 2,
    "project": null,
    "time_finished": null,
    "time_submitted": "2014-03-06T15:40:58.338904",
    "job_id": 1
  }
]
```

```
    }
  ]
}
```

**Statuscode 200** no error

**Statuscode 404** agent not found

**methods** = ['GET', 'POST']

**post** (*agent\_id*)

A POST to this endpoint will assign an existing task to the agent.

**POST** /api/v1/agents/<str:agent\_id>/tasks/ HTTP/1.1

**Request**

```
POST /api/v1/agents/238d7334-8ca5-4469-9f54-e76c66614a43/tasks/ HTTP/1.1
Accept: application/json

{
  "id": 2
}
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "agent_id": 1,
  "parents": [],
  "attempts": 2,
  "children": [],
  "job": {
    "title": "Test Job",
    "id": 1
  },
  "project_id": null,
  "agent": {
    "ip": null,
    "hostname": "agent1",
    "port": 50000,
    "id": "238d7334-8ca5-4469-9f54-e76c66614a43"
  },
  "hidden": false,
  "job_id": 1,
  "time_submitted": "2014-03-06T15:40:58.338904",
  "frame": 2.0,
  "priority": 0,
  "state": "assign",
  "time_finished": null,
  "id": 2,
  "project": null,
  "time_started": null
}
```

**Statuscode 200** no error

**Statuscode 404** agent not found

pyfarm.master.api.agents.**fail\_missing\_assignments** (*agent, current\_assignments*)

pyfarm.master.api.agents.**schema** ()

Returns the basic schema of *Agent*

**GET /api/v1/agents/schema HTTP/1.1**

**Request**

```
GET /api/v1/agents/schema HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "ram": "INTEGER",
  "free_ram": "INTEGER",
  "time_offset": "INTEGER",
  "use_address": "INTEGER",
  "hostname": "VARCHAR(255)",
  "cpus": "INTEGER",
  "port": "INTEGER",
  "state": "INTEGER",
  "ram_allocation": "FLOAT",
  "cpu_allocation": "FLOAT",
  "id": "UUIDType",
  "remote_ip": "IPv4Address"
}
```

**Statuscode 200** no error

**pyfarm.master.api.jobgroups module**

**Job Groups** This module defines an API for managing and querying job groups

**class** pyfarm.master.api.jobgroups.**JobGroupIndexAPI**

Bases: flask.views.MethodView

**get** ()

A GET to this endpoint will return a list of known job groups.

**GET /api/v1/jobgroups/ HTTP/1.1**

**Request**

```
GET /api/v1/jobgroups/ HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": 2,
    "user": "testuser",
```



```

        "main_jobtype": "Test JobType",
        "title": "Test Group"
    }
]

```

**Statuscode 200** no error

**methods** = ['GET', 'POST']

**post** ()

A POST to this endpoint will create a new job group.

**POST** /api/v1/jobgroups/ HTTP/1.1

**Request**

```

POST /api/v1/jobgroups/ HTTP/1.1
Accept: application/json

{
    "title": "Test Group",
    "user": "testuser",
    "main_jobtype": "Test JobType"
}

```

**Response**

```

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

{
    "id": 2,
    "jobs": [],
    "user": "testuser",
    "main_jobtype": "Test JobType",
    "title": "Test Group"
}

```

**Statuscode 201** a new job group was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**class** pyfarm.master.api.jobgroups.**JobsInJobGroupIndexAPI**

**Bases:** flask.views.MethodView

**get** (*group\_id*)

A GET to this endpoint will return all jobs in the speicfied jobgroup.

**GET** /api/v1/jobgroups/<int:id>/jobs HTTP/1.1

**Request**

```

GET /api/v1/jobgroups/2/jobs HTTP/1.1
Accept: application/json

```

**Response**

```

HTTP/1.1 200 OK
Content-Type: application/json

{
  "jobs":
  [
    {
      "id": "12345",
      "title": "Test Job",
      "state": "queued",
      "jobtype_id": 5,
      "jobtype": "Test Jobtype",
      "tasks_queued": 5,
      "tasks_running": 0,
      "tasks_done": 0,
      "tasks_failed": 0
    }
  ]
}

```

**Statuscode 200** no error

**Statuscode 404** the requested job group was not found

**methods = ['GET']**

**class** pyfarm.master.api.jobgroups.**SingleJobGroupAPI**

Bases: flask.views.MethodView

**delete** (*group\_id*)

A DELETE to this endpoint will delete the specified job group

**DELETE** /api/v1/jobgroup/<int:id>

**Request**

```

DELETE /api/v1/jobgroups/1 HTTP/1.1
Accept: application/json

```

**Response**

```

HTTP/1.1 204 NO_CONTENT

```

**Statuscode 204** the job group was deleted or didn't exist

**Statuscode 409** the job group cannot be deleted because it still contains jobs

**get** (*group\_id*)

A GET to this endpoint will return the requested job group

**GET** /api/v1/jobgroups/<int:id> HTTP/1.1

**Request**

```

GET /api/v1/jobgroups/2 HTTP/1.1
Accept: application/json

```

**Response**

```

HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": 2,
  "user": "testuser",
  "main_jobtype": "Test JobType",
  "jobs": [],
  "title": "Test Group"
}

```

**Statuscode 200** no error

**Statuscode 404** the requested job group was not found

**methods** = ['DELETE', 'GET', 'POST']

**post** (*group\_id*)

A POST to this endpoint will update the specified group with the data in the request. Columns not specified in the request will be left as they are.

**POST** /api/v1/jobgroups/<int:id> HTTP/1.1

**Request**

```

POST /api/v1/jobgroups/2 HTTP/1.1
Accept: application/json

{
  "user": "testuser2"
}

```

**Response**

```

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

{
  "id": 2,
  "user": "testuser2",
  "main_jobtype": "Test JobType",
  "jobs": [],
  "title": "Test Group"
}

```

**Statuscode 200** the job group was updated

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

pyfarm.master.api.jobgroups.**schema** ()

Returns the basic schema of *JobGroup*

**GET** /api/v1/jobgroups/schema HTTP/1.1

**Request**

```

GET /api/v1/jobgroups/schema HTTP/1.1
Accept: application/json

```

### Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "main_jobtype": "VARCHAR(64)",
  "title": "VARCHAR(255)",
  "user": "VARCHAR(255)",
  "id": "INTEGER"
}
```

Statuscode 200 no error

## pyfarm.master.api.jobqueues module

**Job Queues** This module defines an API for managing and querying job queues

**class** `pyfarm.master.api.jobqueues.JobQueueIndexAPI`

Bases: `flask.views.MethodView`

**get** ()

A GET to this endpoint will return a list of known job queues.

**GET** `/api/v1/jobqueues/` HTTP/1.1

**Request**

```
GET /api/v1/jobqueues/ HTTP/1.1
Accept: application/json
```

### Response

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "priority": 5,
    "weight": 10,
    "parent_jobqueue_id": null,
    "name": "Test Queue",
    "minimum_agents": null,
    "id": 1,
    "maximum_agents": null
  },
  {
    "priority": 5,
    "weight": 10,
    "parent_jobqueue_id": null,
    "name": "Test Queue 2",
    "minimum_agents": null,
    "id": 2,
    "maximum_agents": null
  }
]
```

Statuscode 200 no error

```
methods = ['GET', 'POST']
```

```
post ()
```

A POST to this endpoint will create a new job queue.

```
POST /api/v1/jobqueues/ HTTP/1.1
```

**Request**

```
POST /api/v1/jobqueues/ HTTP/1.1
Accept: application/json

{
    "name": "Test Queue"
}
```

**Response**

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

{
    "weight": 10,
    "jobs": [],
    "minimum_agents": null,
    "priority": 5,
    "name": "Test Queue",
    "maximum_agents": null,
    "id": 1,
    "parent": null,
    "parent_jobqueue_id": null
}
```

**Statuscode 201** a new job queue was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**Statuscode 409** a job queue with that name already exists

```
class pyfarm.master.api.jobqueues.SingleJobQueueAPI
```

```
Bases: flask.views.MethodView
```

```
delete (queue_rq)
```

A DELETE to this endpoint will delete the specified job queue

```
DELETE /api/v1/jobqueue/ [<str:name>|<int:id>]
```

**Request**

```
DELETE /api/v1/jobqueues/Test%20Queue HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 204 NO_CONTENT
```

**Statuscode 204** the job queue was deleted or didn't exist

**Statuscode 409** the job queue cannot be deleted because it still contains jobs or child queues

**get** (*queue\_rq*)

A GET to this endpoint will return the requested job queue

**GET** /api/v1/jobqueues/[<str:name>|<int:id>] HTTP/1.1

**Request**

```
GET /api/v1/jobqueues/Test%20Queue HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": 1,
  "parent": [],
  "jobs": [],
  "weight": 10,
  "parent_jobqueue_id": null,
  "priority": 5,
  "minimum_agents": null,
  "name": "Test Queue",
  "maximum_agents": null
}
```

**Statuscode 200** no error

**Statuscode 404** the requested job queue was not found

**methods** = ['DELETE', 'GET', 'POST']

**post** (*queue\_rq*)

A POST to this endpoint will update the specified queue with the data in the request. Columns not specified in the request will be left as they are.

**POST** /api/v1/jobqueues/[<str:name>|<int:id>] HTTP/1.1

**Request**

```
POST /api/v1/jobqueues/Test%20Queue HTTP/1.1
Accept: application/json

{
  "priority": 6
}
```

**Response**

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

{
  "id": 1,
  "parent": [],
  "jobs": [],
  "weight": 10,
  "parent_jobqueue_id": null,
  "priority": 6,
}
```

```

    "minimum_agents": null,
    "name": "Test Queue",
    "maximum_agents": null
}

```

**Statuscode 200** the job queue was updated

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

`pyfarm.master.api.jobqueues.schema()`

Returns the basic schema of *JobQueue*

**GET** `/api/v1/jobqueues/schema` HTTP/1.1

**Request**

```

GET /api/v1/jobqueues/schema HTTP/1.1
Accept: application/json

```

**Response**

```

HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "INTEGER",
  "name": "VARCHAR(255)",
  "minimum_agents": "INTEGER",
  "maximum_agents": "INTEGER",
  "priority": "INTEGER",
  "weight": "INTEGER",
  "parent_jobqueue_id": "INTEGER"
}

```

**Statuscode 200** no error

### pyfarm.master.api.jobs module

**Jobs** This module defines an API for managing and querying jobs

**class** `pyfarm.master.api.jobs.JobIndexAPI`

Bases: `flask.views.MethodView`

**get** ()

A GET to this endpoint will return a list of all jobs.

**GET** `/api/v1/jobs/` HTTP/1.1

**Request**

```

GET /api/v1/jobs/ HTTP/1.1
Accept: application/json

```

**Response**

```

HTTP/1.1 200 OK
Content-Type: application/json

```

```
[
  {
    "title": "Test Job",
    "state": "queued",
    "id": 1
  },
  {
    "title": "Test Job 2",
    "state": "queued",
    "id": 2
  }
]
```

Statuscode 200 no error

**methods = ['GET', 'POST']**

**post ()**

A POST to this endpoint will submit a new job.

**POST /api/v1/jobs/ HTTP/1.1**

**Request**

```
POST /api/v1/jobs/ HTTP/1.1
Accept: application/json

{
  "end": 2.0,
  "title": "Test Job 2",
  "jobtype": "TestJobType",
  "data": {
    "foo": "bar"
  },
  "software_requirements": [
    {
      "software": "blender"
    }
  ],
  "start": 1.0
}
```

**Response**

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

{
  "time_finished": null,
  "time_started": null,
  "end": 2.0,
  "time_submitted": "2014-03-06T15:40:58.335259",
  "jobtype_version": 1,
  "jobtype": "TestJobType",
  "jobqueue": None
  "start": 1.0,
  "priority": 0,
  "state": "queued",
  "parents": [],
```



```

    "hidden": false,
    "project_id": null,
    "ram_warning": null,
    "title": "Test Job 2",
    "tags": [],
    "user": null,
    "by": 1.0,
    "data": {
        "foo": "bar"
    },
    "ram_max": null,
    "notes": "",
    "batch": 1,
    "project": null,
    "environ": null,
    "requeue": 3,
    "software_requirements": [
        {
            "min_version": null,
            "max_version": null,
            "max_version_id": null,
            "software_id": 1,
            "min_version_id": null,
            "software": "blender"
        }
    ],
    "tag_requirements": [
        {
            "tag": "workstation",
            "negate": true
        }
    ],
    "id": 2,
    "ram": 32,
    "cpus": 1,
    "children": []
}

```

**Statuscode 201** a new job item was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**Statuscode 404** a referenced object, like a software or software version, does not exist

**Statuscode 409** a conflicting job already exists

**class** `pyfarm.master.api.jobs.JobNotifiedUsersIndexAPI`

Bases: `flask.views.MethodView`

**get** (*job\_name*)

A GET to this endpoint will return a list of all users to be notified on events in this job.

**GET** `/api/v1/jobs/[<str:name>|<int:id>]/notified_users/` HTTP/1.1

**Request**

```

GET /api/v1/jobs/Test%20Job%202/notified_users/ HTTP/1.1
Accept: application/json

```

### Response

```

HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": 1,
    "username": "testuser",
    "email": "testuser@localhost"
  }
]

```

**Statuscode 200** no error

**Statuscode 404** job not found

`methods = ['GET', 'POST']`

`post (job_name)`

A POST to this endpoint will add the specified user to the list of notified users for this job.

**POST** /api/v1/jobs/ [<str:name>|<int:id>] /notified\_users/ HTTP/1.1

**Request**

```

POST /api/v1/jobs/Test%20Job/notified_users/ HTTP/1.1
Accept: application/json

{
  "username": "testuser"
  "on_success": true,
  "on_failure": true,
  "on_deletion": false
}

```

### Response

```

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

{
  "id": 1
  "username": "testuser"
  "email": "testuser@example.com"
}

```

**Statuscode 201** a new notified user entry was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**Statuscode 404** the job or the specified user does not exist

`class pyfarm.master.api.jobs.JobSingleNotifiedUserAPI`

`Bases: flask.views.MethodView`

`delete (job_name, username)`

A DELETE to this endpoint will remove the specified user from the list of notified users for this job.

**DELETE** /api/v1/jobs/[<str:name>|<int:id>]/notified\_users/<str:username> HTTP/1.1  
Request

```
DELETE /api/v1/jobs/Test%20Job/notified_users/testuser HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 204 NO_CONTENT
```

**Statuscode 204** the notified user was removed from this job or wasn't in the list in the first place

**Statuscode 404** the job or the specified user does not exist

**methods** = ['DELETE']

**class** pyfarm.master.api.jobs.JobSingleTaskAPI

Bases: flask.views.MethodView

**get** (*job\_name, task\_id*)

A GET to this endpoint will return the requested task

**GET** /api/v1/jobs/[<str:name>|<int:id>]/tasks/<int:task\_id> HTTP/1.1

**Request**

```
GET /api/v1/jobs/Test%20Job%202/tasks/1 HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "time_finished": null,
  "agent": null,
  "attempts": 0,
  "frame": 2.0,
  "agent_id": null,
  "job": {
    "id": 1,
    "title": "Test Job"
  },
  "time_started": null,
  "state": "running",
  "project_id": null,
  "id": 2,
  "time_submitted": "2014-03-06T15:40:58.338904",
  "project": null,
  "parents": [],
  "job_id": 1,
  "hidden": false,
  "children": [],
  "priority": 0
}
```

**Statuscode 200** no error

`methods = ['GET', 'POST']`

`post` (*job\_name*, *task\_id*)

A POST to this endpoint will update the specified task with the data in the request. Columns not specified in the request will be left as they are. The agent will use this endpoint to inform the master of its progress.

**POST** /api/v1/jobs/ [<str:name>|<int:id> ]/tasks/<int:task\_id> HTTP/1.1

**Request**

```
PUT /api/v1/job/Test%20Job/tasks/1 HTTP/1.1
Accept: application/json

{
    "state": "running"
}
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "time_finished": null,
    "agent": null,
    "attempts": 0,
    "failures": 0,
    "frame": 2.0,
    "agent_id": null,
    "job": {
        "id": 1,
        "title": "Test Job"
    },
    "time_started": null,
    "state": "running",
    "project_id": null,
    "id": 2,
    "time_submitted": "2014-03-06T15:40:58.338904",
    "project": null,
    "parents": [],
    "job_id": 1,
    "hidden": false,
    "children": [],
    "priority": 0
}
```

**Statuscode 200** the task was updated

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**class** `pyfarm.master.api.jobs.JobTasksIndexAPI`

Bases: `flask.views.MethodView`

`get` (*job\_name*)

A GET to this endpoint will return a list of all tasks in a job.

**GET** /api/v1/jobs/ [<str:name>|<int:id> ]/tasks HTTP/1.1

**Request**

```
GET /api/v1/jobs/Test%20Job%202/tasks/ HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "hidden": false,
    "id": 3,
    "attempts": 0,
    "priority": 0,
    "time_started": null,
    "time_submitted": "2014-03-06T15:49:51.892228",
    "frame": 1.0,
    "time_finished": null,
    "job_id": 2,
    "project_id": null,
    "state": "queued",
    "agent_id": null
  },
  {
    "hidden": false,
    "id": 4,
    "attempts": 0,
    "priority": 0,
    "time_started": null,
    "time_submitted": "2014-03-06T15:49:51.892925",
    "frame": 2.0,
    "time_finished": null,
    "job_id": 2,
    "project_id": null,
    "state": "queued",
    "agent_id": null
  }
]
```

**Statuscode 200** no error

```
methods = ['GET']
```

```
exception pyfarm.master.api.jobs.ObjectNotFound
```

```
Bases: Exception
```

```
class pyfarm.master.api.jobs.SingleJobAPI
```

```
Bases: flask.views.MethodView
```

```
delete(job_name)
```

A DELETE to this endpoint will mark the specified job for deletion and remove it after stopping and removing all of its tasks.

```
DELETE /api/v1/jobs/ [<str:name>|<int:id>] HTTP/1.1
```

**Request**

```
DELETE /api/v1/jobs/1 HTTP/1.1
Accept: application/json
```

### Response

```
HTTP/1.1 204 NO_CONTENT
```

**Statuscode 204** the specified job was marked for deletion

**Statuscode 404** the job does not exist

### get (*job\_name*)

A GET to this endpoint will return the specified job, by name or id.

```
GET /api/v1/jobs/[<str:name>|<int:id>] HTTP/1.1
```

### Request

```
GET /api/v1/jobs/Test%20Job%202 HTTP/1.1
Accept: application/json
```

### Response

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "ram_warning": null,
  "title": "Test Job",
  "state": "queued",
  "jobtype_version": 1,
  "jobtype": "TestJobType",
  "environ": null,
  "user": null,
  "priority": 0,
  "time_finished": null,
  "start": 2.0,
  "id": 1,
  "notes": "",
  "notified_users": []
  "ram": 32,
  "tags": [],
  "hidden": false,
  "data": {
    "foo": "bar"
  },
  "software_requirements": [
    {
      "software": "blender",
      "software_id": 1,
      "min_version": null,
      "max_version": null,
      "min_version_id": null,
      "max_version_id": null
    }
  ],
  "tag_requirements": [
    {
```

```

        "tag": "workstation",
        "negate": true
    }
],
"batch": 1,
"time_started": null,
"time_submitted": "2014-03-06T15:40:58.335259",
"requeue": 3,
"end": 4.0,
"parents": [],
"cpus": 1,
"ram_max": null,
"children": [],
"by": 1.0,
"project_id": null
}

```

**Statuscode 200** no error

**Statuscode 404** job not found

**methods** = ['DELETE', 'GET', 'POST']

**post** (*job\_name*)

A POST to this endpoint will update the specified job with the data in the request. Columns not specified in the request will be left as they are. If the “start”, “end” or “by” columns are updated, tasks will be created or deleted as required.

**POST** /api/v1/jobs/ [<str:name>|<int:id>] HTTP/1.1

**Request**

```

PUT /api/v1/jobs/Test%20Job HTTP/1.1
Accept: application/json

{
    "start": 2.0
}

```

**Response**

```

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

{
    "end": 4.0,
    "children": [],
    "jobtype_version": 1,
    "jobtype": "TestJobType",
    "time_started": null,
    "tasks_failed": [],
    "project_id": null,
    "id": 1,
    "software_requirements": [
        {
            "software": "blender",
            "min_version": null,
            "max_version_id": null,
            "software_id": 1,

```

```

        "max_version": null,
        "min_version_id": null
    },
    ],
    "tags": [],
    "environ": null,
    "requeue": 3,
    "start": 2.0,
    "ram_warning": null,
    "title": "Test Job",
    "batch": 1,
    "time_submitted": "2014-03-06T15:40:58.335259",
    "ram_max": null,
    "user": null,
    "notes": "",
    "data": {
        "foo": "bar"
    },
    "tag_requirements": [
        {
            "tag": "workstation",
            "negate": true
        }
    ],
    ],
    "ram": 32,
    "parents": [],
    "hidden": false,
    "priority": 0,
    "cpus": 1,
    "state": "queued",
    "by": 1.0,
    "time_finished": null
}

```

**Statuscode 200** the job was updated

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**class** pyfarm.master.api.jobs.**SingleTaskOnAgentFailureAPI**

Bases: flask.views.MethodView

**delete** (*job\_id, task\_id, agent\_id*)

A DELETE to this endpoint will remove the specified agent from the list of agents that failed to execute this task

**DELETE** /api/v1/jobs/<int:id>/tasks/<int:task\_id>/failed\_on\_agents/<str:agent\_id> HT

**Request**

```
DELETE /api/v1/jobs/12/tasks/1234/failed_on_agents/02f08241-c556-4355-9e5e-33243d8c4577 HT
```

**Response**

```
HTTP/1.1 204 NO_CONTENT
```

**Statuscode 204** the given agent was removed from the list of agents that failed to execute this task



**Statuscode 404** the job, task or agent does not exist

**methods = ['DELETE']**

**class** `pyfarm.master.api.jobs.TaskFailedOnAgentsIndexAPI`

Bases: `flask.views.MethodView`

**get** (*job\_id, task\_id*)

A GET to this endpoint will return a list of all agents that failed to execute this task

**GET** `/api/v1/jobs/<int:job_id>/tasks/<int:task_id>/failed_on_agents/` HTTP/1.1  
Request

```
GET /api/v1/jobs/12/tasks/1234/failed_on_agents/ HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": "02f08241-c556-4355-9e5e-33243d8c4577",
    "hostname": "agent1"
  }
]
```

**Statuscode 200** no error

**Statuscode 404** job or task not found

**methods = ['GET', 'POST']**

**post** (*job\_id, task\_id*)

A POST to this endpoint will add the specified agent to the list of agents that failed to execute this task

**POST** `/api/v1/jobs/<int:id>/tasks/<int:task_id>/failed_on_agents/` HTTP/1.1  
Request

```
POST /api/v1/jobs/12/tasks/1234/failed_on_agents/ HTTP/1.1
Accept: application/json
Content-Type: application/json

{
  "id": "02f08241-c556-4355-9e5e-33243d8c4577"
}
```

**Response**

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

{
  "id": "02f08241-c556-4355-9e5e-33243d8c4577",
  "hostname": "agent1"
}
```

**Statuscode 201** a new entry was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**Statuscode 404** the job, task or agent specified does not exist

`pyfarm.master.api.jobs.parse_requirements` (*requirements*)

Takes a list of dicts specifying a software and optional min- and max-versions and returns a list of `JobRequirement` objects.

Raises `TypeError` if the input was not as expected or `ObjectNotFound` if a referenced software or version was not found.

**Parameters** `requirements` (*list*) – A list of dicts specifying a software and optionally `min_version` and/or `max_version`.

**Raises**

- **TypeError** – Raised if `requirements` is not a list or if an entry in `requirements` is not a dictionary.
- **ValueError** – Raised if there's a problem with the content of at least one of the requirement dictionaries.
- **ObjectNotFound** – Raised if the referenced software or version was not found

`pyfarm.master.api.jobs.schema` ()

Returns the basic schema of `Job`

**GET** `/api/v1/jobs/schema` **HTTP/1.1**

**Request**

`GET /api/v1/jobs/schema` **HTTP/1.1**  
`Accept: application/json`

**Response**

```

HTTP/1.1 200 OK
Content-Type: application/json

{
  "batch": "INTEGER",
  "by": "NUMERIC(10, 4)",
  "cpus": "INTEGER",
  "data": "JSONDict",
  "end": "NUMERIC(10,4)",
  "environ": "JSONDict",
  "hidden": "BOOLEAN",
  "id": "INTEGER",
  "jobtype": "VARCHAR(64)",
  "jobtype_version": "INTEGER",
  "jobqueue": "VARCHAR(255)",
  "notes": "TEXT",
  "priority": "INTEGER",
  "project_id": "INTEGER",
  "ram": "INTEGER",
  "ram_max": "INTEGER",
  "ram_warning": "INTEGER",
  "requeue": "INTEGER",
  "start": "NUMERIC(10,4)",
  "state": "WorkStateEnum",

```

```

    "time_finished": "DATETIME",
    "time_started": "DATETIME",
    "time_submitted": "DATETIME",
    "title": "VARCHAR(255)",
    "user": "VARCHAR(255)"
}

```

**Statuscode 200** no error

### pyfarm.master.api.jobtypes module

**Jobtypes** This module defines an API for managing and querying jobtypes

**class** pyfarm.master.api.jobtypes.**JobTypeCodeAPI**

Bases: flask.views.MethodView

**get** (*jobtype\_name, version*)

A GET to this endpoint will return just the python code for this version of the specified jobtype.

**GET** /api/v1/jobtypes/[<str:name>|<int:id>]/versions/<int:version>/code HTTP/1.1  
Request

```

GET /api/v1/jobtypes/TestJobType/versions/1/code HTTP/1.1
Accept: text/x-python

```

#### Response

```

HTTP/1.1 200 OK
Content-Type: text/x-python

from pyfarm.jobtypes.core.jobtype import JobType

class TestJobType(JobType):
    def get_command(self):
        return "/usr/bin/touch"

    def get_arguments(self):
        return [os.path.join(
            self.assignment_data["job"]["data"]["path"], "%04d" %
            self.assignment_data["tasks"][0]["frame"])]

```

**Statuscode 200** no error

**Statuscode 404** jobtype or version not found

**methods** = ['GET']

**class** pyfarm.master.api.jobtypes.**JobTypeIndexAPI**

Bases: flask.views.MethodView

**get** ()

A GET to this endpoint will return a list of registered jobtypes.

**GET** /api/v1/jobtypes/ HTTP/1.1  
Request

```
GET /api/v1/jobtypes/ HTTP/1.1
Accept: application/json
```

### Response

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": 1,
    "name": "TestJobType"
  }
]
```

**Statuscode 200** no error

**methods = ['GET', 'POST']**

**post ()**

A POST to this endpoint will create a new jobtype.

**POST /api/v1/jobtypes/ HTTP/1.1**

### Request

```
POST /api/v1/jobtypes/ HTTP/1.1
Accept: application/json

{
  "name": "TestJobType",
  "classname": "TestJobType",
  "description": "Jobtype for testing inserts and queries",
  "code": "\nfrom pyfarm.jobtypes.core.jobtype import "
        "JobType\n\nclass TestJobType(JobType):\n"
        "    def get_command(self):\n"
        "        return "/usr/bin/touch"\n\n"
        "    def get_arguments(self):\n"
        "        return [os.path.join("
        "self.assignment_data["job"]["data"]["path"], "
        "%04d" % self.assignment_data["tasks"]"
        "[0]["frame"])]\n"
}
```

### Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": 1,
  "batch_contiguous": true,
  "software_requirements": [],
  "version": 1,
  "max_batch": 1,
  "name": "TestJobType",
  "classname": "TestJobType",
  "description": "Jobtype for testing inserts and queries",
```

```

"code": "\nfrom pyfarm.jobtypes.core.jobtype import "
        "JobType\n\nclass TestJobType(JobType):\n"
        "    def get_command(self):\n"
        "        return "/usr/bin/touch"\n\n"
        "    def get_arguments(self):\n"
        "        return [os.path.join("
        "self.assignment_data["job"]["data"]["path"], "
        "\"%04d" % self.assignment_data["tasks"]"
        "[0]["frame"])]\n"
    }

```

**Statuscode 201** a new jobtype item was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**Statuscode 409** a conflicting jobtype already exists

**class** `pyfarm.master.api.jobtypes.JobTypeSoftwareRequirementAPI`

Bases: `flask.views.MethodView`

**delete** (*jobtype\_name*, *software*)

A DELETE to this endpoint will delete the requested software requirement from the specified jobtype, creating a new version of the jobtype in the process

**DELETE** `/api/v1/jobtypes/<str:name>|<int:id>/software_requirements/<int:id>` HTTP/1.1

**Request**

```

DELETE /api/v1/jobtypes/TestJobType/software_requirements/1 HTTP/1.1
Accept: application/json

```

**Response**

```

HTTP/1.1 204 NO CONTENT

```

**Statuscode 204** the software requirement was deleted or didn't exist

**get** (*jobtype\_name*, *software*)

A GET to this endpoint will return the specified software requirement from the newest version of the requested jobtype.

**GET** `/api/v1/jobtypes/<str:name>|<int:id>/software_requirements/<int:id>` HTTP/1.1

**Request**

```

GET /api/v1/jobtypes/TestJobType/software_requirements/1 HTTP/1.1
Accept: application/json

```

**Response**

```

HTTP/1.1 200 OK
Content-Type: application/json

{
    "software": {
        "software": "/bin/touch",
        "id": 1
    },

```

```
    "max_version": null,
    "min_version": {
        "version": "8.21",
        "id": 1
    },
    "jobtype_version": {
        "version": 7,
        "jobtype": "TestJobType"
    }
}
```

**Statuscode 200** no error

**Statuscode 404** jobtype or software requirement not found

`methods = ['DELETE', 'GET']`

`class pyfarm.master.api.jobtypes.JobTypeSoftwareRequirementsIndexAPI`

`Bases: flask.views.MethodView`

`get (jobtype_name, version=None)`

A GET to this endpoint will return a list of all the software requirements of the specified jobtype

**GET** /api/v1/jobtypes/[<str:name>|<int:id>]/software\_requirements/ HTTP/1.1

**Request**

```
GET /api/v1/jobtypes/TestJobType/software_requirements/ HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "software": {
      "software": "/bin/touch",
      "id": 1
    },
    "max_version": null,
    "min_version": {
      "version": "8.21",
      "id": 1
    },
    "jobtype_version": {
      "version": 7,
      "jobtype": "TestJobType"
    }
  }
]
```

**Statuscode 200** no error

**Statuscode 404** jobtype or version not found

`methods = ['GET', 'POST']`

**post** (*jobtype\_name*, *version=None*)

A POST to this endpoint will create a new software\_requirement for the specified jobtype. This will transparently create a new jobtype version

**POST** /api/v1/jobtypes/[<str:name>|<int:id>]/software\_requirements/ HTTP/1.1  
**Request**

```
POST /api/v1/jobtypes/TestJobType/software_requirements/ HTTP/1.1
Accept: application/json

{
  "software": "blender",
  "min_version": "2.69"
}
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "jobtype_version": {
    "id": 8,
    "jobtype": "TestJobType",
    "version": 7
  },
  "max_version": null,
  "min_version": {
    "id": 2,
    "version": "1.69"
  },
  "software": {
    "id": 2,
    "software": "blender"
  }
}
```

**Statuscode 201** a new software requirement was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**Statuscode 405** you tried calling this method on a specific version

**Statuscode 409** a conflicting software requirement already exists

**class** pyfarm.master.api.jobtypes.**JobTypeVersionsIndexAPI**

Bases: flask.views.MethodView

**get** (*jobtype\_name*)

A GET to this endpoint will return a sorted list of of all known versions of the specified jobtype.

**GET** /api/v1/jobtypes/[<str:name>|<int:id>]/versions/ HTTP/1.1  
**Request**

```
GET /api/v1/jobtypes/TestJobType/versions/ HTTP/1.1
Accept: application/json
```

### Response

```
HTTP/1.1 200 OK
Content-Type: application/json

[1, 2]
```

**Statuscode 200** no error

**Statuscode 404** jobtype not found

**methods** = ['GET']

**exception** pyfarm.master.api.jobtypes.**ObjectNotFound**

Bases: `Exception`

**class** pyfarm.master.api.jobtypes.**SingleJobTypeAPI**

Bases: `flask.views.MethodView`

**delete** (*jobtype\_name*)

A DELETE to this endpoint will delete the requested jobtype

**DELETE** /api/v1/jobtypes/[<str:name>|<int:id>] HTTP/1.1

### Request

```
DELETE /api/v1/jobtypes/TestJobType HTTP/1.1
Accept: application/json
```

### Response

```
HTTP/1.1 204 NO CONTENT
```

**Statuscode 204** the jobtype was deleted or didn't exist

**get** (*jobtype\_name*)

A GET to this endpoint will return the most recent version of the referenced jobtype, by name or id.

**GET** /api/v1/jobtypes/<str:tagname> HTTP/1.1

### Request

```
GET /api/v1/jobtypes/TestJobType HTTP/1.1
Accept: application/json
```

### Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "batch_contiguous": true,
  "classname": null,
  "code": "\nfrom pyfarm.jobtypes.core.jobtype import "
        "JobType\n\nclass TestJobType(JobType):\n"
        "    def get_command(self):\n"
        "        return "/usr/bin/touch"\n\n"
        "    def get_arguments(self):\n"
        "        return [os.path.join("

```



```

        "self.assignment_data["job"]["data"]["path"], "
        "%04d" % self.assignment_data["tasks"]
        "[0] ["frame"])]\n",
    "id": 1,
    "version": 1,
    "max_batch": 1,
    "name": "TestJobType",
    "software_requirements": [
        {
            "max_version": null,
            "max_version_id": null,
            "min_version": "8.21",
            "min_version_id": 1,
            "software": "/bin/touch",
            "software_id": 1
        }
    ]
}

```

**Statuscode 200** no error

**Statuscode 404** jobtype or version not found

**methods** = ['DELETE', 'GET', 'PUT']

**put** (*jobtype\_name*)

A PUT to this endpoint will create a new jobtype under the given URI. If a jobtype already exists under that URI, a new version will be created with the given data.

You should only call this by id for updating an existing jobtype or if you have a reserved jobtype id. There is currently no way to reserve a jobtype id.

**PUT /api/v1/jobtypes/[<str:name>|<int:id>] HTTP/1.1**

**Request**

```

PUT /api/v1/jobtypes/TestJobType HTTP/1.1
Accept: application/json

{
    "name": "TestJobType",
    "description": "Jobtype for testing inserts and queries",
    "code": "\nfrom pyfarm.jobtypes.core.jobtype import "
            "JobType\n\nclass TestJobType(JobType):\n"
            "    def get_command(self):\n"
            "        return "/usr/bin/touch"\n\n"
            "    def get_arguments(self):\n"
            "        return [os.path.join("
    "self.assignment_data["job"]["data"]["path"], "
    "%04d" % self.assignment_data["tasks"]
    "[0] ["frame"])]\n"
}

```

**Response**

```

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

{

```

```

    "batch_contiguous": true,
    "classname": null,
    "code": "\nfrom pyfarm.jobtypes.core.jobtype import "
            "JobType\n\nclass TestJobType(JobType):\n"
            "    def get_command(self):\n"
            "        return "/usr/bin/touch"\n\n"
            "    def get_arguments(self):\n"
            "        return [os.path.join("
            "self.assignment_data["job"]["data"]["path"], "
            ""%04d" % self.assignment_data["tasks"]"
            "[0]["frame"])]\n",
    "id": 1,
    "max_batch": 1,
    "name": "TestJobType",
    "description": "Jobtype for testing inserts and queries",
    "software_requirements": []
}

```

**Statuscode 201** a new jobtype was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**class** pyfarm.master.api.jobtypes.**VersionedJobTypeAPI**

Bases: flask.views.MethodView

**delete** (*jobtype\_name*, *version*)

A DELETE to this endpoint will delete the requested version of the specified jobtype.

**DELETE** /api/v1/jobtypes/[<str:name>|<int:id>]/versions/<int:version> HTTP/1.1  
Request

```

DELETE /api/v1/jobtypes/TestJobType/versions/1 HTTP/1.1
Accept: application/json

```

**Response**

```

HTTP/1.1 204 NO CONTENT

```

**Statuscode 204** the version was deleted or didn't exist

**get** (*jobtype\_name*, *version*)

A GET to this endpoint will return the specified version of the referenced jobtype, by name or id.

**GET** /api/v1/jobtypes/[<str:name>|<int:id>]/versions/<int:version> HTTP/1.1  
Request

```

GET /api/v1/jobtypes/TestJobType/versions/1 HTTP/1.1
Accept: application/json

```

**Response**

```

HTTP/1.1 200 OK
Content-Type: application/json

{
    "batch_contiguous": true,

```

```

"classname": null,
"name": "TestJobType",
"code": "\nfrom pyfarm.jobtypes.core.jobtype import "
        "JobType\n\nclass TestJobType(JobType):\n"
        "    def get_command(self):\n"
        "        return "/usr/bin/touch"\n\n"
        "    def get_arguments(self):\n"
        "        return [os.path.join("
        "self.assignment_data["job"]["data"]["path"], "
        ""%04d" % self.assignment_data["tasks"]"
        "[0]["frame"])]\n",
"id": 1,
"version": 1,
"max_batch": 1,
"software_requirements": [
    {
        "max_version": null,
        "max_version_id": null,
        "min_version": "8.21",
        "min_version_id": 1,
        "software": "/bin/touch",
        "software_id": 1
    }
]
}

```

**Statuscode 200** no error

**Statuscode 404** jobtype or version not found

**methods** = ['DELETE', 'GET']

pyfarm.master.api.jobtypes.**parse\_requirements** (*requirements*)

Takes a list of dicts specifying a software and optional min- and max-versions and returns a list of JobRequirement objects.

Raises TypeError if the input was not as expected or ObjectNotFound if a referenced software or version was not found.

**Parameters requirements** (*list*) – A list of dicts specifying a software and optionally min\_version and/or max\_version.

**Raises**

- **TypeError** – Raised if requirements is not a list or if an entry in requirements is not a dictionary.
- **ValueError** – Raised if there's a problem with the content of at least one of the requirement dictionaries.
- **ObjectNotFound** – Raised if the referenced software or version was not found

pyfarm.master.api.jobtypes.**schema** ()

Returns the basic schema of *JobType*

**GET** /api/v1/jobtypes/schema HTTP/1.1

**Request**

```

GET /api/v1/jobtypes/schema HTTP/1.1
Accept: application/json

```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "batch_contiguous": "BOOLEAN",
  "classname": "VARCHAR(64)",
  "code": "TEXT",
  "description": "TEXT",
  "id": "INTEGER",
  "version": "INTEGER",
  "max_batch": "INTEGER",
  "no_automatic_start_time": "INTEGER",
  "name": "VARCHAR(64)"
}
```

Statuscode 200 no error

**pyfarm.master.api.pathmaps module**

**Path Maps** API endpoints for viewing and managing path maps

**class** pyfarm.master.api.pathmaps.**PathMapIndexAPI**

Bases: flask.views.MethodView

**get** ()

A GET to this endpoint will return a list of all registered path maps, with id. It can be made with a for\_agent query parameter, in which case it will return only those path maps that apply to that agent.

**GET** /api/v1/pathmaps/ HTTP/1.1

**Request**

```
GET /api/v1/pathmaps/ HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": 1,
    "path_osx": "/mnt/nfs",
    "path_windows": "\\domains\cifs_server",
    "path_linux": "/mnt/nfs"
  },
  {
    "id": 7,
    "path_osx": "/renderout",
    "path_windows": "c:\renderout",
    "path_linux": "/renderout"
  }
]
```

**Statuscode 200** no error

`methods = ['GET', 'POST']`

`post ()`

A POST to this endpoint will create a new path map.

A path map will list the equivalent path prefixes for all three supported families of operating systems, Linux, Windows and OS X. A path map can optionally be restricted to one tag, in which case it will only apply to agents with that tag. If a tag is specified that does not exist yet, that tag will be transparently created.

**POST** `/api/v1/pathmaps/` HTTP/1.1

**Request**

```
POST /api/v1/pathmaps/ HTTP/1.1
Accept: application/json

{
  "path_linux": "/mnt/nfs",
  "path_windows": "\\domain\\cifs_server",
  "path_osx": "/mnt/nfs",
  "tag": "production"
}
```

**Response**

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

{
  "id": 1,
  "path_linux": "/mnt/nfs",
  "path_windows": "\\domain\\cifs_server",
  "path_osx": "/mnt/nfs",
  "tag": "production"
}
```

**Statuscode 201** a new pathmap was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

`class pyfarm.master.api.pathmaps.SinglePathMapAPI`

`Bases: flask.views.MethodView`

`delete (pathmap_id)`

A DELETE to this endpoint will remove the specified pathmap

**DELETE** `/api/v1/pathmaps/<int:pathmap_id>` HTTP/1.1

**Request**

```
DELETE /api/v1/pathmaps/1 HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 204 NO_CONTENT
```

**Statuscode 204** the path map was deleted or did not exist in the first place

**get** (*pathmap\_id*)

A GET to this endpoint will return a single path map specified by *pathmap\_id*

**GET** /api/v1/pathmaps/<int:pathmap\_id> HTTP/1.1

**Request**

```
GET /api/v1/pathmaps/1 HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "id": 1,
  "path_osx": "/mnt/nfs",
  "path_windows": "\\domains\cifs_server",
  "path_linux": "/mnt/nfs"
}
```

**Statuscode 200** no error

**methods** = ['DELETE', 'GET', 'POST']

**post** (*pathmap\_id*)

A POST to this endpoint will update an existing path map with new values.

Only the values included in the request will be updated. The rest will be left unchanged. The id column cannot be changed. Including it in the request will lead to an error.

**POST** /api/v1/pathmaps/<int:pathmap\_id> HTTP/1.1

**Request**

```
POST /api/v1/pathmaps/1 HTTP/1.1
Accept: application/json
```

```
{
  "path_linux": "/mnt/smb"
}
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
  "id": 1,
  "path_linux": "/mnt/smb",
  "path_windows": "\\domain\cifs_server",
  "path_osx": "/mnt/nfs",
}
```

```

    "tag": "production"
  }

```

**Statuscode 200** the specified pathmap was updated

**Statuscode 404** the specified pathmap does not exist

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

`pyfarm.master.api.pathmaps.schema()`

Returns the basic schema of *Agent*

**GET** `/api/v1/pathmaps/schema` HTTP/1.1

**Request**

```

GET /api/v1/pathmaps/schema HTTP/1.1
Accept: application/json

```

**Response**

```

HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "INTEGER",
  "path_linux": "VARCHAR(512)",
  "path_windows": "VARCHAR(512)",
  "path_osx": "VARCHAR(512)",
  "tag": "VARCHAR(64)"
}

```

**Statuscode 200** no error

### pyfarm.master.api.software module

**Software** Contained within this module are an API handling functions which can manage or query software items using JSON.

**class** `pyfarm.master.api.software.SingleSoftwareAPI`

Bases: `flask.views.MethodView`

**delete** (*software\_rq*)

A DELETE to this endpoint will delete the requested software tag

**DELETE** `/api/v1/software/<str:softwarename>` HTTP/1.1

**Request**

```

DELETE /api/v1/software/Autodesk%20Maya HTTP/1.1
Accept: application/json

```

**Response**

```

HTTP/1.1 204 NO_CONTENT

```

**Statuscode 204** the software tag was deleted or didn't exist

**get** (*software\_rq*)

A GET to this endpoint will return the requested software tag

**GET** /api/v1/software/<str:softwarename> HTTP/1.1

**Request**

```
GET /api/v1/software/Autodesk%20Maya HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "software": "Autodesk Maya",
  "id": 1,
  "versions": [
    {
      "version": "2013",
      "id": 1,
      "rank": 100
    },
    {
      "version": "2014",
      "id": 2,
      "rank": 200
    }
  ]
}
```

**Statuscode 200** no error

**Statuscode 404** the requested software tag was not found

**methods** = ['DELETE', 'GET', 'PUT']

**put** (*software\_rq*)

A PUT to this endpoint will create a new software tag under the given URI or update an existing software tag if one exists. Renaming existing software tags via this call is supported, but when creating new ones, the included software name must be equal to the one in the URI.

You should only call this by id for overwriting an existing software tag or if you have a reserved software id. There is currently no way to reserve a tag id.

**PUT** /api/v1/software/<str:softwarename> HTTP/1.1

**Request**

```
PUT /api/v1/software/blender HTTP/1.1
Accept: application/json

{
  "software": "blender"
}
```

**Response**



```

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

{
  "id": 4,
  "software": "blender",
  "versions": []
}

```

**Request**

```

PUT /api/v1/software/blender HTTP/1.1
Accept: application/json

{
  "software": "blender",
  "version": [
    {
      "version": "1.69"
    }
  ]
}

```

**Response**

```

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

{
  "id": 4,
  "software": "blender",
  "versions": [
    {
      "version": "1.69",
      "id": 1,
      "rank": 100
    }
  ]
}

```

**Statuscode 200** an existing software tag was updated

**Statuscode 201** a new software tag was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**class** `pyfarm.master.api.software.SingleSoftwareVersionAPI`

Bases: `flask.views.MethodView`

**delete** (*software\_rq*, *version\_name*)

A DELETE to this endpoint will delete the requested software version

**DELETE** `/api/v1/software/<str:softwarename>/versions/<str:version>` HTTP/1.1

**Request**

```
DELETE /api/v1/software/Autodesk%20Maya/versions/2013 HTTP/1.1
Accept: application/json
```

#### Response

```
HTTP/1.1 204 NO_CONTENT
```

**Statuscode 204** the software version was deleted or didn't exist

**Statuscode 404** the software specified does not exist

**get** (*software\_rq*, *version\_name*)

A GET to this endpoint will return the specified version

**GET** /api/v1/software/<str:softwarename>/versions/<str:version> HTTP/1.1  
**Request**

```
GET /api/v1/software/Autodesk%20Maya/versions/2014 HTTP/1.1
Accept: application/json
```

#### Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "version": "2013",
  "id": 1,
  "rank": 100,
  "discovery_function_name": null
}
```

**Statuscode 200** no error

**Statuscode 404** the requested software tag or version was not found

```
methods = ['DELETE', 'GET']
```

```
class pyfarm.master.api.software.SoftwareIndexAPI
  Bases: flask.views.MethodView
```

**get** ()

A GET to this endpoint will return a list of known software, with all known versions.

**GET** /api/v1/software/ HTTP/1.1  
**Request**

```
GET /api/v1/software/ HTTP/1.1
Accept: application/json
```

#### Response

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
```

```

    "software": "Houdini",
    "id": 1,
    "versions": [
      {
        "version": "13.0.1",
        "id": 1,
        "rank": 100
      }
    ]
  }
]

```

**Statuscode 200** no error

**methods** = ['GET', 'POST']

**post** ()

A POST to this endpoint will create a new software tag.

A list of versions can be included. If the software item already exists the listed versions will be added to the existing ones. Versions with no explicit rank are assumed to be the newest version available. Users should not mix versions with an explicit rank with versions without one.

**POST /api/v1/software/ HTTP/1.1**

**Request**

```

POST /api/v1/software/ HTTP/1.1
Accept: application/json

{
  "software": "blender"
}

```

**Response (new software item create)**

```

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

{
  "id": 4,
  "software": "blender",
  "versions": []
}

```

**Statuscode 201** a new software item was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**Statuscode 409** a software tag with that name already exists

**class** pyfarm.master.api.software.**SoftwareVersionDiscoveryCodeAPI**

**Bases:** flask.views.MethodView

**get** (*software\_rq*, *version\_name*)

A GET to this endpoint will return just the python code for detecting whether this software version is installed on an agent.

**GET** /api/v1/software/[<str:software\_name>|<int:software\_id>]/versions/<str:version>  
**Request**

```
GET /api/v1/software/Blender/versions/2.72/code HTTP/1.1
Accept: text/x-python
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: text/x-python

def blender_2_72_installed()
    return True
```

**Statuscode 200** no error

**Statuscode 404** software or version not found or this software version has no discovery code defined

**methods = ['GET']**

**class** pyfarm.master.api.software.**SoftwareVersionsIndexAPI**

Bases: flask.views.MethodView

**get** (*software\_rq*)

A GET to this endpoint will list all known versions for this software

**GET** /api/v1/software/<str:softwarename>/versions/ HTTP/1.1

**Request**

```
GET /api/v1/software/Autodesk%20Maya/versions/ HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "version": "2013",
    "id": 1,
    "rank": 100
  },
  {
    "version": "2014",
    "id": 2,
    "rank": 200
  }
]
```

**Statuscode 200** no error

**Statuscode 404** the requested software tag was not found

**methods = ['GET', 'POST']**

**post** (*software\_rq*)

A POST to this endpoint will create a new version for this software.

A rank can optionally be included. If it isn't, it is assumed that this is the newest version for this software

**POST** /api/v1/software/versions/ HTTP/1.1

**Request**

```
POST /api/v1/software/blender/versions/ HTTP/1.1
Accept: application/json

{
  "version": "1.70"
}
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": 4,
  "version": "1.70",
  "rank": "100"
}
```

**Statuscode 201** a new software version was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**Statuscode 409** a software version with that name already exists

**exception** `pyfarm.master.api.software.VersionParseError`

Bases: `Exception`

Raised by `extract_version_dicts()` when the function is unable to parse a version.

`pyfarm.master.api.software.extract_version_dicts(json_in)`

Extracts and returns a list of versions from `json_in`.

`pyfarm.master.api.software.schema()`

Returns the basic schema of `Software`

**GET** /api/v1/software/schema HTTP/1.1

**Request**

```
GET /api/v1/software/schema HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "INTEGER",
  "software": "VARCHAR(64)"
}
```

Statuscode 200 no error

### pyfarm.master.api.tags module

**Tag** Contained within this module are an API handling functions which can manage or query tags using JSON.

**class** pyfarm.master.api.tags.**AgentsInTagIndexAPI**

Bases: flask.views.MethodView

**get** (tagname=None)

A GET to this endpoint will list all agents associated with this tag.

**GET** /api/v1/tags/<str:tagname>/agents/ HTTP/1.1  
**Request**

```
GET /api/v1/tags/interesting/agents/ HTTP/1.1
Accept: application/json
```

#### Response

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

[
  {
    "hostname": "agent3",
    "id": 1,
    "href": "/api/v1/agents/1"
  }
]
```

**Statuscode 200** the list of agents associated with this tag is returned

**Statuscode 404** the tag specified does not exist

**methods** = ['GET', 'POST']

**post** (tagname=None)

A POST will add an agent to the list of agents tagged with this tag The tag can be given as a string or as an integer (its id).

**POST** /api/v1/tags/<str:tagname>/agents/ HTTP/1.1  
**Request**

```
POST /api/v1/tags/interesting/agents/ HTTP/1.1
Accept: application/json

{
  "agent_id": "dd0c6da2-0c91-42cf-a82f-6d503aae43d3"
}
```

#### Response (agent newly tagged)

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

{
```

```

    "href": "/api/v1/agents/1",
    "id": 1
  }

```

**Request**

```

POST /api/v1/tags/interesting/agents/ HTTP/1.1
Accept: application/json

{
  "agent_id": "dd0c6da2-0c91-42cf-a82f-6d503aae43d3"
}

```

**Response (agent already had that tag)**

```

HTTP/1.1 200 OK
Content-Type: application/json

{
  "href": "/api/v1/agents/1",
  "id": 1
}

```

**Statuscode 200** an existing tag was found and returned

**Statuscode 201** a new tag was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**Statuscode 404** either the tag or the referenced agent does not exist

```
class pyfarm.master.api.tags.SingleTagAPI
```

```
  Bases: flask.views.MethodView
```

```
  delete (tagname=None)
```

A DELETE to this endpoint will delete the tag under this URI, including all relations to tags or jobs.

```
  DELETE /api/v1/tags/<str:tagname> HTTP/1.1
```

**Request**

```

DELETE /api/v1/tags/interesting HTTP/1.1
Accept: application/json

```

**Response**

```

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

{
  "id": 1,
  "tag": "interesting"
}

```

**Statuscode 204** the tag was deleted or did not exist in the first place

**get** (*tagname=None*)

A GET to this endpoint will return the referenced tag, either by name or id, including a list of agents and jobs associated with it.

**GET /api/v1/tags/<str:tagname> HTTP/1.1**  
**Request**

```
GET /api/v1/tags/interesting HTTP/1.1
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "agents": [{
    "hostname": "agent3",
    "href": "/api/v1/agents/94522b7e-817b-4358-95da-670b31aad624",
    "id": 1
  }],
  "id": 1,
  "jobs": [],
  "tag": "interesting"
}
```

**Statuscode 200** no error

**Statuscode 404** tag not found

**methods = ['DELETE', 'GET', 'PUT']**

**put** (*tagname=None*)

A PUT to this endpoint will create a new tag under the given URI. If a tag already exists under that URI, it will be deleted, then recreated. Note that when overwriting a tag like that, all relations that are not explicitly specified here will be deleted. You can optionally specify a list of agents or jobs relations as integers in the request data.

You should only call this by id for overwriting an existing tag or if you have a reserved tag id. There is currently no way to reserve a tag id.

**PUT /api/v1/tags/<str:tagname> HTTP/1.1**  
**Request**

```
PUT /api/v1/tags/interesting HTTP/1.1
Accept: application/json

{
  "tag": "interesting"
}
```

**Response**

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

{
  "id": 1,
```



```

    "tag": "interesting"
  }

```

**Request**

```

PUT /api/v1/tags/interesting HTTP/1.1
Accept: application/json

{
  "tag": "interesting",
  "agents": [1]
  "jobs": []
}

```

**Response**

```

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

{
  "id": 1,
  "tag": "interesting"
}

```

**Statuscode 201** a new tag was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**Statuscode 404** a referenced agent or job does not exist

**class** pyfarm.master.api.tags.**TagIndexAPI**

Bases: flask.views.MethodView

**get** ()

A GET to this endpoint will return a list of known tags, with id. Associated agents and jobs are included for every tag

:rtype: object .. http:get:: /api/v1/tags/ HTTP/1.1

**Request**

```

GET /api/v1/tags/ HTTP/1.1
Accept: application/json

```

**Response**

```

HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "agents": [
      1
    ],
    "jobs": [],
    "id": 1,
    "tag": "interesting"
  }
]

```

```
    },
    {
      "agents": [],
      "jobs": [],
      "id": 2,
      "tag": "boring"
    }
  ]
```

**Statuscode 200** no error

`methods = ['GET', 'POST']`

`post ()`

A POST to this endpoint will do one of two things:

- create a new tag and return the row
- return the row for an existing tag

Tags only have one column, the tag name. Two tags are automatically considered equal if the tag names are equal.

**POST /api/v1/tags/ HTTP/1.1**

**Request**

```
POST /api/v1/tags/ HTTP/1.1
Accept: application/json

{
  "tag": "interesting"
}
```

**Response (new tag create)**

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

{
  "id": 1,
  "tag": "interesting"
}
```

**Request**

```
POST /api/v1/tags/ HTTP/1.1
Accept: application/json

{
  "tag": "interesting"
}
```

**Response (existing tag returned)**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
```

```

    "id": 1,
    "tag": "interesting"
}

```

**Statuscode 200** an existing tag was found and returned

**Statuscode 201** a new tag was created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

`pyfarm.master.api.tags.schema()`

Returns the basic schema of *Tag*

**GET** `/api/v1/tags/schema/` **HTTP/1.1**

**Request**

```

GET /api/v1/tags/schema/ HTTP/1.1
Accept: application/json

```

**Response**

```

HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": "INTEGER",
  "tag": "VARCHAR(64)"
}

```

**Statuscode 200** no error

### pyfarm.master.api.tasklogs module

**Task Logs** This module defines an API for managing and querying logs belonging to tasks

**class** `pyfarm.master.api.tasklogs.LogsInTaskAttemptsIndexAPI`

Bases: `flask.views.MethodView`

**get** (*job\_id, task\_id, attempt*)

A GET to this endpoint will return a list of all known logs that are associated with this attempt at running this task

**GET** `/api/v1/jobs/<job_id>/tasks/<task_id>/attempts/<attempt>/logs/` **HTTP/1.1**

**Request**

```

GET /api/v1/jobs/4/tasks/1300/attempts/5/logs/ HTTP/1.1
Accept: application/json

```

**Response**

```

HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "agent_id": "3087ada4-290a-45b0-8c1a-21db4cd284fc",

```

```

        "created_on": "2014-09-03T10:58:59.754880",
        "identifier": "2014-09-03_10-58-59_4_4ee02475335911e4a935c86000cbf5fb.csv"
    }
]

```

**Statuscode 200** no error

**Statuscode 404** the specified task was not found

**methods** = ['GET', 'POST']

**post** (*job\_id, task\_id, attempt*)

A POST to this endpoint will register a new logfile with the given attempt at running the given task

A logfile has an identifier which must be unique in the system. If two tasks get assigned a logfile with the same id, it is considered to be the same log.

**POST** /api/v1/jobs/<job\_id>/tasks/<task\_id>/attempts/<attempt>/logs/ HTTP/1.1  
**Request**

```

POST /api/v1/jobs/4/tasks/1300/attempts/5/logs/ HTTP/1.1
Content-Type: application/json

{
    "identifier": "2014-09-03_10-58-59_4_4ee02475335911e4a935c86000cbf5fb.csv",
    "agent_id": "2dc2cb5a-35da-41d6-8864-329c0d7d5391"
}

```

**Response**

```

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

{
    "identifier": "2014-09-03_10-58-59_4_4ee02475335911e4a935c86000cbf5fb.csv",
    "agent_id": "2dc2cb5a-35da-41d6-8864-329c0d7d5391",
    "created_on": "2014-09-03T10:59:05.103005",
    "id": 148
}

```

**Statuscode 201** the association between this task attempt and logfile has been created

**Statuscode 400** there was something wrong with the request (such as invalid columns being included)

**Statuscode 404** the specified task does not exist

**Statuscode 409** the specified log was already registered on the specified task

**class** pyfarm.master.api.tasklogs.**SingleLogInTaskAttempt**

Bases: flask.views.MethodView

**get** (*job\_id, task\_id, attempt, log\_identifier*)

A GET to this endpoint will return metadata about the specified logfile

**GET** /api/v1/jobs/<job\_id>/tasks/<task\_id>/attempts/<attempt>/logs/<log\_identifier>  
**Request**

```
GET /api/v1/jobs/4/tasks/1300/attempts/5/logs/2014-09-03_10-58-59_4_4ee02475335911e4a935
Accept: application/json
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": 147,
  "identifier": "2014-09-03_10-58-59_4_4ee02475335911e4a935c86000cbf5fb.csv",
  "created_on": "2014-09-03T10:58:59.754880",
  "agent_id": "836ce137-6ad4-443f-abb9-94c4465ff87c"
}
```

**Statuscode 200** no error**Statuscode 404** task or logfile not found**methods = ['GET', 'POST']****post** (*job\_id*, *task\_id*, *attempt*, *log\_identifier*)

A POST to this endpoint will update metadata about the specified logfile

**POST** /api/v1/jobs/<job\_id>/tasks/<task\_id>/attempts/<attempt>/logs/<log\_identifier>**Request**

```
POST /api/v1/jobs/4/tasks/1300/attempts/5/logs/2014-09-03_10-58-59_4_4ee02475335911e4a935
Accept: application/json
Content-Type: application/json

{
  "state": "done"
}
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "id": 147,
  "identifier": "2014-09-03_10-58-59_4_4ee02475335911e4a935c86000cbf5fb.csv",
  "created_on": "2014-09-03T10:58:59.754880",
  "agent_id": "836ce137-6ad4-443f-abb9-94c4465ff87c"
}
```

**Statuscode 200** no error**Statuscode 404** task or logfile not found**class** pyfarm.master.api.tasklogs.**TaskLogfileAPI**

Bases: flask.views.MethodView

**get** (*job\_id*, *task\_id*, *attempt*, *log\_identifier*)

A GET to this endpoint will return the actual logfile or a redirect to it.

**GET** /api/v1/jobs/<job\_id>/tasks/<task\_id>/attempts/<attempt>/logs/<log\_identifier>/  
**Request**

```
GET /api/v1/jobs/4/tasks/1300/attempts/5/logs/2014-09-03_10-58-59_4_4ee02475335911e4a935/
Accept: text/csv
```

**Response**

```
HTTP/1.1 200 OK
Content-Type: text/csv

<Content of the logfile>
```

**Statuscode 200** no error

**Statuscode 307** The logfile can be found in another location at this point in time. Independent future requests for the same logfile should continue using the original URL

**Statuscode 400** the specified logfile identifier is not acceptable

**Statuscode 404** task or logfile not found

**methods** = ['GET', 'PUT']

**put** (*job\_id*, *task\_id*, *attempt*, *log\_identifier*)

A PUT to this endpoint will upload the request's body as the specified logfile

**PUT** /api/v1/jobs/<job\_id>/tasks/<task\_id>/attempts/<attempt>/logs/<log\_identifier>/  
**Request**

```
PUT /api/v1/jobs/4/tasks/1300/attempts/5/logs/2014-09-03_10-58-59_4_4ee02475335911e4a935/
<content of the logfile>
```

**Response**

```
HTTP/1.1 201 CREATED
```

**Statuscode 201** logfile was uploaded

**Statuscode 400** the specified logfile identifier is not acceptable

**Statuscode 404** task or logfile not found

## Module contents

### 3.1.2 pyfarm.master.user\_interface package

#### Subpackages

pyfarm.master.user\_interface.statistics package

#### Submodules

### pyfarm.master.user\_interface.statistics.agent\_counts module

pyfarm.master.user\_interface.statistics.agent\_counts.**agent\_counts**()

### pyfarm.master.user\_interface.statistics.index module

pyfarm.master.user\_interface.statistics.index.**statistics\_index**()

### pyfarm.master.user\_interface.statistics.task\_events module

class pyfarm.master.user\_interface.statistics.task\_events.**TotalsAverage**(*first\_sample*,  
*last\_sum=None*)

Bases: object

**add\_sample**(*sample*)

**avg\_done**()

**avg\_failed**()

**avg\_queued**()

**avg\_running**()

pyfarm.master.user\_interface.statistics.task\_events.**task\_events**()

## Module contents

### Submodules

#### pyfarm.master.user\_interface.agents module

pyfarm.master.user\_interface.agents.**agent\_add\_software**(*agent\_id*)

pyfarm.master.user\_interface.agents.**agent\_delete\_software**(*agent\_id*, *version\_id*)

pyfarm.master.user\_interface.agents.**agents**()

pyfarm.master.user\_interface.agents.**check\_software\_in\_single\_agent**(*agent\_id*)

pyfarm.master.user\_interface.agents.**delete\_multiple\_agents**()

pyfarm.master.user\_interface.agents.**delete\_single\_agent**(*agent\_id*)

pyfarm.master.user\_interface.agents.**disable\_multiple\_agents**()

pyfarm.master.user\_interface.agents.**disable\_single\_agent**(*agent\_id*)

pyfarm.master.user\_interface.agents.**enable\_multiple\_agents**()

pyfarm.master.user\_interface.agents.**enable\_single\_agent**(*agent\_id*)

pyfarm.master.user\_interface.agents.**restart\_multiple\_agents**()

pyfarm.master.user\_interface.agents.**restart\_single\_agent**(*agent\_id*)

pyfarm.master.user\_interface.agents.**single\_agent**(*agent\_id*)

pyfarm.master.user\_interface.agents.**update\_notes\_for\_agent**(*agent\_id*)

pyfarm.master.user\_interface.agents.**update\_tags\_in\_agent**(*agent\_id*)

### pyfarm.master.user\_interface.jobgroups module

pyfarm.master.user\_interface.jobgroups.**jobgroups** ()

### pyfarm.master.user\_interface.jobqueues module

pyfarm.master.user\_interface.jobqueues.**delete\_jobqueue** (*queue\_id*)

pyfarm.master.user\_interface.jobqueues.**delete\_subqueue** (*queue*)

pyfarm.master.user\_interface.jobqueues.**jobqueue** (*queue\_id*)

pyfarm.master.user\_interface.jobqueues.**jobqueue\_create** ()

pyfarm.master.user\_interface.jobqueues.**jobqueues** ()

### pyfarm.master.user\_interface.jobs module

pyfarm.master.user\_interface.jobs.**add\_notified\_user\_to\_job** (*job\_id*)

pyfarm.master.user\_interface.jobs.**add\_tag\_on\_jobs** ()

pyfarm.master.user\_interface.jobs.**add\_tag\_requirement\_on\_jobs** ()

pyfarm.master.user\_interface.jobs.**alter\_autodeletion\_for\_job** (*job\_id*)

pyfarm.master.user\_interface.jobs.**alter\_frames\_in\_single\_job** (*job\_id*)

pyfarm.master.user\_interface.jobs.**alter\_scheduling\_parameters\_for\_job** (*job\_id*)

pyfarm.master.user\_interface.jobs.**delete\_multiple\_jobs** ()

pyfarm.master.user\_interface.jobs.**delete\_single\_job** (*job\_id*)

pyfarm.master.user\_interface.jobs.**jobs** ()

pyfarm.master.user\_interface.jobs.**move\_multiple\_jobs** ()

pyfarm.master.user\_interface.jobs.**pause\_multiple\_jobs** ()

pyfarm.master.user\_interface.jobs.**pause\_single\_job** (*job\_id*)

pyfarm.master.user\_interface.jobs.**remove\_notified\_user\_from\_job** (*job\_id*,  
*user\_id*)

pyfarm.master.user\_interface.jobs.**remove\_tag\_from\_jobs** ()

pyfarm.master.user\_interface.jobs.**remove\_tag\_requirement\_from\_jobs** ()

pyfarm.master.user\_interface.jobs.**rerun\_failed\_in\_job** (*job\_id*)

pyfarm.master.user\_interface.jobs.**rerun\_failed\_in\_multiple\_jobs** ()

pyfarm.master.user\_interface.jobs.**rerun\_multiple\_jobs** ()

pyfarm.master.user\_interface.jobs.**rerun\_single\_job** (*job\_id*)

pyfarm.master.user\_interface.jobs.**rerun\_single\_task** (*job\_id*, *task\_id*)

pyfarm.master.user\_interface.jobs.**set\_prio\_weight\_on\_jobs** ()

pyfarm.master.user\_interface.jobs.**single\_job** (*job\_id*)

pyfarm.master.user\_interface.jobs.**unpause\_multiple\_jobs** ()



```

pyfarm.master.user_interface.jobs.unpause_single_job(job_id)
pyfarm.master.user_interface.jobs.update_notes_for_job(job_id)
pyfarm.master.user_interface.jobs.update_tag_requirements_in_job(job_id)
pyfarm.master.user_interface.jobs.update_tags_in_job(job_id)
pyfarm.master.user_interface.jobs.upgrade_job_to_latest_jobtype_version(job_id)

```

### pyfarm.master.user\_interface.jobtypes module

**Jobtypes** UI endpoints allowing seeing and manipulating jobtypes via the web interface

```

pyfarm.master.user_interface.jobtypes.add_jobtype_software_requirement(jobtype_id)
pyfarm.master.user_interface.jobtypes.create_jobtype()
pyfarm.master.user_interface.jobtypes.jobtype(jobtype_id)
    UI endpoint for a single jobtype. Allows showing and updating the jobtype
pyfarm.master.user_interface.jobtypes.jobtypes()
pyfarm.master.user_interface.jobtypes.remove_jobtype(jobtype_id)
pyfarm.master.user_interface.jobtypes.remove_jobtype_software_requirement(jobtype_id,
                                                                              soft-
                                                                              ware_id)
pyfarm.master.user_interface.jobtypes.update_jobtype_notification_templates(jobtype_id)

```

### pyfarm.master.user\_interface.software module

```

pyfarm.master.user_interface.software.add_software()
pyfarm.master.user_interface.software.add_software_version(software_id)
pyfarm.master.user_interface.software.remove_software(software_id)
pyfarm.master.user_interface.software.remove_software_version(software_id, ver-
                                                                sion_id)
pyfarm.master.user_interface.software.software()
pyfarm.master.user_interface.software.software_item(software_id)
pyfarm.master.user_interface.software.update_version_default_status(software_id,
                                                                      ver-
                                                                      sion_id)
pyfarm.master.user_interface.software.update_version_rank(software_id, ver-
                                                            sion_id)

```

### pyfarm.master.user\_interface.software\_version module

```

pyfarm.master.user_interface.software_version.software_version(software_id,
                                                                version_id)

```

## Module contents

# 3.2 Submodules

## 3.2.1 pyfarm.master.application module

### Application

Contains the functions necessary to construct the application layer classes necessary to run the master.

**class** `pyfarm.master.application.SessionMixin`

Bases: `object`

Mixin which adds a `_session` attribute. This class is provided mainly to limit issues with circular imports.

**class** `pyfarm.master.application.UUIDConverter` (*map*)

Bases: `werkzeug.routing.BaseConverter`

A URL converter for UUIDs. This class is loaded as part of the Flask application setup and may be used in url routing:

```
@app.route('/foo/<uuid:value>')
def foobar(value):
    pass
```

When a request such as `GET /foo/F9A63B47-66BF-4E2B-A545-879986BB7CA9` is made `UUIDConverter` will receive value to `to_python()` which will then convert the string to an instance of `UUID`.

`to_python` (*value*)

`to_url` (*value*)

`pyfarm.master.application.before_request` ()

Global `before_request` handler that will handle common problems when trying to accept json data to the api.

`pyfarm.master.application.get_api_blueprint` (*url\_prefix=None*)

Constructs and returns an instance of `Blueprint` for routing api requests.

**Parameters** `url_prefix` (*string*) – The url prefix for the api such as `/api/v1`. If not provided then value will be derived from the `api_prefix` configuration variable.

`pyfarm.master.application.get_application` (\*\**configuration\_keywords*)

Returns a new application context. If keys and values are provided to `config_values` they will be used to override the default configuration values or create new ones

```
>>> app = get_application(TESTING=True)
>>> assert app.testing is True
```

**Parameters** `setup_appcontext` (*bool*) – If `True` then setup the `flask.g` variable to include the application level information (ex. `g.db`)

`pyfarm.master.application.get_login_manager` (\*\**kwargs*)

Constructs and returns an instance of `LoginManager`. Any keyword arguments provided will be passed to the constructor of `LoginManager`

`pyfarm.master.application.get_login_serializer` (*secret\_key*)

Constructs and returns and instance of `URLSafeTimedSerializer`

`pyfarm.master.application.get_sqlalchemy` (*app=None, use\_native\_unicode=True, session\_options=None*)  
 Constructs and returns an instance of `SQLAlchemy`. Any keyword arguments provided will be passed to the constructor of `SQLAlchemy`

## 3.2.2 pyfarm.master.config module

### Configuration

A small wrapper around `pyfarm.core.config.Configuration` that loads in the configuration files and provides backwards compatibility for some environment variables.

**class** `pyfarm.master.config.Configuration`  
 Bases: `pyfarm.core.config.Configuration`

The main configuration object for the master, models and scheduler. This will load in the configuration files and also handle any overrides present in the environment.

**Variables** `ENVIRONMENT_OVERRIDES` – A dictionary containing all environment variables we support as overrides. This set is mainly provided for backwards comparability purposes or for the rare case where an environment override would be preferred over a config.

`ENVIRONMENT_OVERRIDES = {'secret_key': ('PYFARM_SECRET_KEY', functools.partial(<function read_env at 0x7f...`

## 3.2.3 pyfarm.master.entrypoints module

### Entry Points

Contains the code which operates the Python entry point scripts as well as serving as a central location for the construction of the web application.

`pyfarm.master.entrypoints.create_app()`  
 An entry point specifically for uWSGI or similar to use

`pyfarm.master.entrypoints.load_api` (*app\_instance, api\_instance*)  
 configures flask to serve the api endpoints

`pyfarm.master.entrypoints.load_authentication` (*app\_instance*)  
 configures flask to serve the authentication endpoints

`pyfarm.master.entrypoints.load_before_first` (*app\_instance, database\_instance*)

`pyfarm.master.entrypoints.load_error_handlers` (*app\_instance*)  
 loads the error handlers onto application instance

`pyfarm.master.entrypoints.load_index` (*app\_instance*)  
 configures flask to serve the main index and favicon

`pyfarm.master.entrypoints.load_master` (*app, api*)  
 loads and attaches all endpoints needed to run the master

`pyfarm.master.entrypoints.load_setup` (*app\_instance*)  
 configures flask to serve the endpoint used for setting up the system

`pyfarm.master.entrypoints.load_user_interface` (*app\_instance*)

`pyfarm.master.entrypoints.run_master()`  
 Runs `load_master()` then runs the application

`pyfarm.master.entrypoints.tables()`

Small script for basic table management and, eventually, some introspection as well.

## 3.2.4 pyfarm.master.index module

### Index

Contains the endpoints for master's index ("/")

`pyfarm.master.index.favicon()`

Sends out the favicon from the static directory

**Warning:** On deployment, /favicon.ico should really be handled by the frontend server and **not** the application.

`pyfarm.master.index.index_page()`

## 3.2.5 pyfarm.master.initial module

### Initial Setup

Entry points for the /setup/ target

`class pyfarm.master.initial.NewUserForm(formdata=None, obj=None, prefix='', data=None, meta=None, **kwargs)`

Bases: `wtforms.form.Form`

`email = <UnboundField(TextField, 0), {'validators': [<wtforms.validators.Required object at 0x7f3d00e057f0>]}>`

`password = <UnboundField>PasswordField, 0), {'validators': [<wtforms.validators.Required object at 0x7f3d00e05828>]}>`

`username = <UnboundField(TextField, 0), {'validators': [<wtforms.validators.Required object at 0x7f3d00a16cc0>]}>`

`validate_username (field)`

`pyfarm.master.initial.setup_page()`

## 3.2.6 pyfarm.master.testutil module

### Test Utilities

Functions and classes mainly used during the unittests.

`class pyfarm.master.testutil.BaseTestCase(methodName='runTest')`

Bases: `unittest.case.TestCase`

`ENVIRONMENT_SETUP = False`

`ORIGINAL_ENVIRONMENT = {'SSH_TTY': '/dev/pts/0', 'LS_COLORS': 'rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=0`

`assert_accepted (response)`

`assert_bad_request (response)`

`assert_conflict (response)`

**assert\_contents\_equal** (*a\_source*, *b\_source*)

Explicitly check to see if the two iterable objects contain the same data. This method exists to check to make sure two iterables contain the same data without regards to order. This is mostly meant for cases where two lists contain unhashable types.

**assert\_created** (*response*)

**assert\_forbidden** (*response*)

**assert\_internal\_server\_error** (*response*)

**assert\_method\_not\_allowed** (*response*)

**assert\_no\_content** (*response*)

**assert\_not\_acceptable** (*response*)

**assert\_not\_found** (*response*)

**assert\_ok** (*response*)

**assert\_status** (*response*, *status\_code=None*)

**assert\_temporary\_redirect** (*response*)

**assert\_unauthorized** (*response*)

**assert\_unsupported\_media\_type** (*response*)

**classmethod build\_environment** ()

Sets up the current environment with some values for unittesting. This must be used before any other code is imported otherwise

**Warning:** This classmethod should not be used outside of a testing context

**maxDiff = None**

**setUp** ()

**setup\_app** ()

Constructs the application object and assigns the instance variables for tests. If you're testing the master your subclass will probably need to extend this method.

**setup\_client** (*app*)

returns the test client from the given application instance

**setup\_database** ()

**setup\_warning\_filter** ()

**tearDown** ()

**teardown\_app** ()

**teardown\_database** ()

**teardown\_warning\_filter** ()

**class** `pyfarm.master.testutil.JsonResponseMixin`

Bases: `object`

Mixin with testing helper methods

**json**

`pyfarm.master.testutil.make_test_response` (*response\_class=None*)

## 3.2.7 pyfarm.master.utility module

### Utility

General utility which are not view or tool specific

```
class pyfarm.master.utility.JSONEncoder (skipkeys=False, ensure_ascii=True,
                                         check_circular=True, allow_nan=True,
                                         sort_keys=False, indent=None, separators=None,
                                         default=None)
```

Bases: `json.encoder.JSONEncoder`

**default** (*o*)

```
pyfarm.master.utility.assert_mimetypes (flask_request, mimetypes)
```

**Warning:** This function will produce an unhandled error if you use it outside of a request.

Check to make sure that the request's mimetype is in `mimetypes`. If this is not true then call `flask.abort()` with `UNSUPPORTED_MEDIA_TYPE`

#### Parameters

- **flask\_request** – The flask request object which we should check the `mimetype` attribute on.
- **mimetypes** (*list, tuple, set*) – The `mimetypes` which `flask_request` can be.

```
pyfarm.master.utility.default_json_encoder (obj)
```

```
pyfarm.master.utility.dumps (obj, **kwargs)
```

Wrapper for `json.dumps()` that ensures `JSONEncoder` is passed in.

```
pyfarm.master.utility.error_handler (e, code=None, default=None, title=None,
                                     template=None)
```

Constructor for http errors that respects the current mimetype. By default this function returns html however when `request.mimetype` is `application/json` it will return a json response. This function is typically used within a `functools.partial()` call:

```
>>> from functools import partial
>>> try:
...     from httplib import BAD_REQUEST
... except ImportError:
...     from http.client import BAD_REQUEST
...
>>> from flask import request
>>> error_400 = partial(
...     error_handler, BAD_REQUEST,
...     lambda: "bad request to %s" % request.url, "Bad Request")
```

#### Parameters

- **e** (*flask.Response*) – The response object which will be passed into `error_handler()`, this value is ignored by default.
- **code** (*int*) – The integer to use in the response. For the most consistent results you can use the `httplib` or `http.client` modules depending on your Python version.
- **default** (*callable*) – This will be the default error message if `g.error` does not contain anything. `default` may either be a callable function which will produce the string or it may be a string by itself.

- **title** (*str*) – The HTML title of the request being made. This is not used when dealing with json requests and if not provided at all will default to using the official status code's string representation.
- **template** (*str*) – A alternative template path for HTML responses

`pyfarm.master.utility.get_g(attribute, instance_types, unset=<object object>)`

Returns data from `flask.g` after checking to make sure the attribute was set and that it has the correct type.

This function does not check to see if you're already inside a request.

#### Parameters

- **attribute** (*str*) – The name of the attribute on the `flask.g` object
- **instance\_types** (*tuple*) – A tuple of classes which the data we're looking for should be a part of

`pyfarm.master.utility.get_request_argument(argument, default=None, required=False, types=None)`

This is a function similar to Flask's `request.args.get` except it does type validation and it has the concept of required url arguments.

#### Parameters

- **argument** (*str*) – The name of the url argument we're trying to retrieve
- **default** – The value to return if `argument` is not present in the url and argument is not a required parameter.
- **required** (*bool*) – If True and the url argument provided by `argument` is not provided respond to the request with `BAD_REQUEST`
- **types** – A single or list of multiple callable objects which will be used to try and produce a result to return. This would function similarly to this:

```
value = "5"
types = (int, bool)

for type_callable in types:
    try:
        return type_callable(value)
    except Exception:
        continue
```

`pyfarm.master.utility.inside_request()`

Returns True if we're inside a request, False if not.

`pyfarm.master.utility.isuuid(value)`

Returns True if `value` is a UUID object or can be converted to one

`pyfarm.master.utility.jsonify(*args, **kwargs)`

Drop in replacement for `flask jsonify()` that also handles list objects as well as a few custom objects like `Decimal` or `datetime`. Flask does not support lists by default because it's considered a security risk in most cases but we do need it in certain cases. Since flask's `jsonify` does not allow passing arbitrary arguments to `json.dumps()`, we cannot use it if the output data contains custom types.

`pyfarm.master.utility.timedelta_format(value)`

Formats a `timedelta` object without the microseconds

`pyfarm.master.utility.validate_json` (*validator*, *json\_types*=(<class 'dict'>,))

A decorator, similar to `validate_with_model()`, but greatly simplified and more flexible. Unlike `validate_with_model()` this decorator is meant to handle data which may not be structured for a model.

#### Parameters

- **mimetype** (*tuple*) – A tuple of mimetypes that are allowed to be handled by the decorated function.
- **json\_types** (*tuple*) – The root type or types which the object on `g.json` should be an instance of.

`pyfarm.master.utility.validate_with_model` (*model*, *type\_checks*=None, *ignore*=None, *ignore\_missing*=None, *disallow*=None)

Decorator which will check the contents of the of the json request against a model for:

- missing fields which are required
- values which don't match their type(s) in the database
- inclusion of fields which do not exist

#### Parameters

- **model** – The model object that the decorated endpoint should use for testing the points above.
- **type\_checks** (*dict*) – A dictionary containing a mapping of column names to special functions used for checking. If there's a key in the incoming request that needs a more detailed check than “`isinstance(g.json[column_name], <Python type(s) from sql>`” then this is the place to add it.
- **ignore\_missing** (*list*) – A list of fields to completely ignore in the incoming request. Typically this is used by PUT requests or other similar requests where part of the data is in the url.
- **allow\_missing** (*list*) – A list of fields which are allowed to be missing in the request. These fields will still be checked for type however.
- **disallow** (*list*) – A list of columns which are never in the request to the decorated function

## 3.3 Module contents

Contains all the necessary code to operate an instance of the master.



---

## pyfarm.models package

---

### 4.1 Subpackages

#### 4.1.1 pyfarm.models.core package

##### Submodules

##### pyfarm.models.core.functions module

**Functions** Contains core functions and data for use by *pyfarm.models*

`pyfarm.models.core.functions.getuuid` (*value*, *table*, *table\_attrib*, *error\_tail*)

Returns the proper value for the given input. Depending on the type being provided this will return one of the following:

- None
- the value from an attribute
- string from a UUID
- the original value (after validating it's a UUID)

##### Parameters

- **value** (*string*) – the value to validate and returning data from
- **table** (*string*) – the table which the provided *value* belongs to
- **table\_attrib** (*string*) – the attribute to use when attempting to pull data off of a model object
- **error\_tail** (*string*) – added to the end of error messages
- **error\_text** (*str*) – error text to render in the event of problems

**Raises** **ValueError** raised when the provided input is invalid, blank, or otherwise unexpected

`pyfarm.models.core.functions.modelfor` (*model*, *table*)

Returns True if the given *model* object is for the expected *table*.

```
>>> from pyfarm.master.config import config
>>> from pyfarm.models.agent import Agent
>>> modelfor(Agent("foo", "10.56.0.0", "255.0.0.0"), config.get("table_agent"))
True
```

`pyfarm.models.core.functions.repr_enum(value, enum=None)`  
produces the string representation of an enum value

`pyfarm.models.core.functions.repr_ip(value)`  
properly formats an `IPAddress` object

`pyfarm.models.core.functions.split_and_extend(items)`  
Takes a list of input elements and splits them before producing an extended set.

#### Example

```
>>> split_and_extend(["root.admin", "admin"])
set(['admin', 'root.admin', 'root'])
```

`pyfarm.models.core.functions.work_columns(state_default, priority_default)`  
Produces some default columns which are used by models which produce work.

### pyfarm.models.core.mixins module

**Mixin Classes** Module containing mixins which can be used by multiple models.

`class pyfarm.models.core.mixins.ModelTypes(primary_keys, autoincrementing, columns, required, relationships, mappings)`

Bases: `tuple`

#### **autoincrementing**

Alias for field number 1

#### **columns**

Alias for field number 2

#### **mappings**

Alias for field number 5

#### **primary\_keys**

Alias for field number 0

#### **relationships**

Alias for field number 4

#### **required**

Alias for field number 3

`class pyfarm.models.core.mixins.ReprMixin`

Bases: `object`

Mixin which allows model classes to to convert columns into a more easily read object format.

#### Variables

- `REPR_COLUMNS` (*tuple*) – the columns to convert
- `REPR_CONVERT_COLUMN` (*dict*) – optional dictionary containing columns names and functions for converting to a more readable string format

`REPR_COLUMNS = NotImplemented`

`REPR_CONVERT_COLUMN = {}`

`class pyfarm.models.core.mixins.UtilityMixins`

Bases: `object`

Mixins which can be used to produce dictionaries of existing data.

**Const dict** `DICT_CONVERT_COLUMN` A dictionary containing key value pairs of attribute names and a function to retrieve the attribute. The function should take a single input and return the value itself. Optionally, you can also use the `NotImplemented` object to exclude some columns from the results.

```
DICT_CONVERT_COLUMN = {}
```

```
to_dict (unpack_relationships=True)
```

Produce a dictionary of existing data in the table

**Parameters** `unpack_relationships` (*list, tuple, set, bool*) – If `True` then unpack all relationships. If `unpack_relationships` is an iterable such as a list or tuple object then only unpack those relationships.

```
classmethod to_schema ()
```

Produce a dictionary which represents the table's schema in a basic format

```
classmethod types ()
```

A classmethod that constructs a `namedtuple` object with four attributes:

- `primary_keys` - set of all primary key(s) names
- `autoincrementing` - set of all columns which have autoincrement set
- `columns` - set of all column names
- `required` - set of all required columns (non-nullable wo/defaults)
- `relationships` - not columns themselves but do store relationships
- `mappings` - contains a dictionary with each field mapping to a Python type

```
class pyfarm.models.core.mixins.ValidatePriorityMixin
```

Bases: `object`

Mixin that adds a `state` column and uses a class level `STATE_ENUM` attribute to assist in validation.

```
MAX_PRIORITY = 1000
```

```
MIN_PRIORITY = -1000
```

```
validate_attempts (key, value)
```

ensures the number of attempts provided is valid

```
validate_priority (key, value)
```

ensures the value provided to priority is valid

```
class pyfarm.models.core.mixins.ValidateWorkStateMixin
```

Bases: `object`

```
STATE_ENUM = NotImplemented
```

```
validate_state (key, value)
```

Ensures that `value` is a member of `STATE_ENUM`

```
validate_state_column (key, value)
```

validates the state column

```
class pyfarm.models.core.mixins.WorkStateChangedMixin
```

Bases: `object`

Mixin which adds a static method to be used when the model state changes

```
static state_changed (target, new_value, old_value, initiator)
```

update the datetime objects depending on the new value

## pyfarm.models.core.types module

**Custom Columns and Type Generators** Special column types used by PyFarm's models.

**class** `pyfarm.models.core.types.AgentStateEnum(*args, **kwargs)`

Bases: `pyfarm.models.core.types.EnumType`

custom column type for working with AgentState

**enum** = `AgentState(ONLINE=Values(202, 'online'), DISABLED=Values(200, 'disabled'), RUNNING=Values(203, 'running'))`

**class** `pyfarm.models.core.types.EnumType(*args, **kwargs)`

Bases: `sqlalchemy.sql.type_api.TypeDecorator`

Special column type which handles translation from a human readable enum into an integer that the database can use.

**Variables** `enum` – required class level variable which defines what enum this custom column handles

**Raises** `AssertionError` raised if `enum` is not set on the class

**enum** = `NotImplemented`

**impl**

alias of `Integer`

**json\_types** = (`<class 'str'>`, `<class 'int'>`)

**process\_bind\_param** (`value, dialect`)

Takes `value` and maps it to the internal integer.

**Raises** `ValueError` raised if `value` is not part of the class level enum mapping

**process\_result\_value** (`value, dialect`)

`pyfarm.models.core.types.IDTypeAgent`

alias of `UUIDType`

**class** `pyfarm.models.core.types.IPAddress(addr, version=None, flags=0)`

Bases: `netaddr.ip.IPAddress`

Custom version of `netaddr.IPAddress` which can match itself against other instance of the same class, a string, or an integer.

**class** `pyfarm.models.core.types.Ipv4Address(*args, **kwargs)`

Bases: `sqlalchemy.sql.type_api.TypeDecorator`

Column type which can store and retrieve IPv4 addresses in a more efficient manner

**MAX\_INT** = `4294967295`

**checkInteger** (`value`)

**impl**

alias of `BigInteger`

**json\_types** = (`<class 'str'>`, `<class 'int'>`)

**process\_bind\_param** (`value, dialect`)

**process\_result\_value** (`value, dialect`)

**class** `pyfarm.models.core.types.JSONDict(*args, **kwargs)`

Bases: `pyfarm.models.core.types.JSONSerializable`

Column type for storing dictionary objects as json

**json\_types**  
alias of `dict`

**serialize\_types** = (<class 'dict'>, <class 'collections.UserDict'>)

**class** `pyfarm.models.core.types.JSONList` (\*args, \*\*kwargs)  
Bases: `pyfarm.models.core.types.JSONSerializable`

Column type for storing list objects as json

**json\_types**  
alias of `list`

**serialize\_types** = (<class 'list'>, <class 'tuple'>, <class 'collections.UserList'>)

**class** `pyfarm.models.core.types.JSONSerializable` (\*args, \*\*kwargs)  
Bases: `sqlalchemy.sql.type_api.TypeDecorator`

Base of all custom types which process json data to and from the database.

#### Variables

- **serialize\_types** (*tuple*) – the kinds of objects we expect to serialize to and from the database
- **serialize\_none** (*bool*) – if True then return None instead of converting it to its json value
- **allow\_blank** (*bool*) – if True, do not raise a `ValueError` for empty data
- **allow\_empty** (*bool*) – if True, do not raise `ValueError` if the input data itself is empty

**dumps** (*value*)

Performs the process of dumping *value* to json. For classes such as `UserDict` or `UserList` this will dump the underlying data instead of the object itself.

**impl**  
alias of `UnicodeText`

**process\_bind\_param** (*value, dialect*)  
Converts the value being assigned into a json blob

**process\_result\_value** (*value, dialect*)  
Converts data from the database into a Python object

**serialize\_none** = `False`

**serialize\_types** = `None`

**class** `pyfarm.models.core.types.MACAddress` (\*args, \*\*kwargs)  
Bases: `sqlalchemy.sql.type_api.TypeDecorator`

Column type which can store and retrieve MAC addresses in a more efficient manner

**MAX\_INT** = `281474976710655`

**impl**  
alias of `BigInteger`

**json\_types** = (<class 'str'>, <class 'int'>)

**process\_bind\_param** (*value, dialect*)

**process\_result\_value** (*value, dialect*)

**class** `pyfarm.models.core.types.OperatingSystemEnum` (\*args, \*\*kwargs)  
 Bases: `pyfarm.models.core.types.EnumType`

custom column type for working with AgentState

**enum** = `OperatingSystem(WINDOWS=Values(301, 'windows'), MAC=Values(302, 'mac'), LINUX=Values(300, 'linux'), ...)`

**class** `pyfarm.models.core.types.UUIDType` (\*args, \*\*kwargs)  
 Bases: `sqlalchemy.sql.type_api.TypeDecorator`

Custom column type which handles UUIDs in the appropriate manner for various databases.

**impl**  
 alias of `TypeEngine`

**json\_types**  
 alias of `UUID`

**load\_dialect\_impl** (*dialect*)

**process\_bind\_param** (*value, dialect*)

**process\_result\_value** (*value, dialect*)

**class** `pyfarm.models.core.types.UseAgentAddressEnum` (\*args, \*\*kwargs)  
 Bases: `pyfarm.models.core.types.EnumType`

custom column type for working with UseAgentAddress

**enum** = `UseAgentAddress(REMOTE=Values(311, 'remote'), HOSTNAME=Values(312, 'hostname'), LOCAL=Values(310, 'local'))`

**class** `pyfarm.models.core.types.WorkStateEnum` (\*args, \*\*kwargs)  
 Bases: `pyfarm.models.core.types.EnumType`

custom column type for working with WorkState

**enum** = `WorkState(FAILED=Values(107, 'failed'), PAUSED=Values(100, 'paused'), RUNNING=Values(105, 'running'), ...)`

`pyfarm.models.core.types.id_column` (*column\_type=None, \*\*kwargs*)

Produces a column used for *id* on each table. Typically this is done using a class in `pyfarm.models.mixins` however because of the ORM and the table relationships it's cleaner to have a function produce the column.

## Module contents

### 4.1.2 pyfarm.models.statistics package

#### Submodules

##### `pyfarm.models.statistics.agent_count` module

**AgentCount Model** Model describing the counts for agents in various states at a given point in time.

**class** `pyfarm.models.statistics.agent_count.AgentCount` (\*\*kwargs)  
 Bases: `flask_sqlalchemy.Model`

**counted\_time**  
 The point in time at which these counts were done

**num\_disabled**  
 The number of agents that were in state *disabled* at *counted\_time*

**num\_offline**

The number of agents that were in state *offline* at *counted\_time*

**num\_online**

The number of agents that were in state *online* at *counted\_time*

**num\_running**

The number of agents that were in state *running* at *counted\_time*

**pyfarm.models.statistics.task\_count module**

**TaskCount Model** Model describing the number of tasks in a given queue in a given state at a point in time

```
class pyfarm.models.statistics.task_count.TaskCount (**kwargs)
```

Bases: flask\_sqlalchemy.Model

**counted\_time**

The point in time at which these counts were done

**id**

Provides an id for the current row. This value should never be directly relied upon and it's intended for use by relationships.

**job\_queue\_id**

ID of the jobqueue these stats refer to

**total\_done**

Number of done tasks at *counted\_time*

**total\_failed**

Number of failed tasks at *counted\_time*

**total\_queued**

Number of queued tasks at *counted\_time*

**total\_running**

Number of running tasks at *counted\_time*

**pyfarm.models.statistics.task\_event\_count module**

**TaskEventCount Model** Model describing the number of events that happened for tasks over a time period

```
class pyfarm.models.statistics.task_event_count.TaskEventCount (**kwargs)
```

Bases: flask\_sqlalchemy.Model

**id**

Provides an id for the current row. This value should never be directly relied upon and it's intended for use by relationships.

**job\_queue\_id**

ID of the jobqueue these stats refer to

**num\_deleted**

Number of tasks that were deleted during the time period

**num\_done**

Number of tasks that were finished successfully during the time period

**num\_failed**

Number of tasks that failed during the time period

**num\_new**  
 Number of tasks that were newly created during the time period

**num\_restarted**  
 Number of tasks that were restarted during the time period

**num\_started**  
 Number of tasks that work was started on during the time period

**time\_end**

**time\_start**

## Module contents

Contains models specifically for gathering runtime statistics about the farm

## 4.2 Submodules

### 4.2.1 pyfarm.models.agent module

#### Agent Models

Models and interface classes related to the agent.

```
class pyfarm.models.agent.Agent(**kwargs)
    Bases: flask_sqlalchemy.Model, pyfarm.models.core.mixins.ValidatePriorityMixin,
    pyfarm.models.core.mixins.ValidateWorkStateMixin, pyfarm.models.core.mixins.UtilityMixin,
    pyfarm.models.core.mixins.ReprMixin
```

Stores information about an agent include its network address, state, allocation configuration, etc.

---

**Note:** This table enforces two forms of uniqueness. The *id* column must be unique and the combination of these columns must also be unique to limit the frequency of duplicate data:

- *hostname*
- *port*
- *id*

---

**MAX\_CPUS = 256**

**MAX\_PORT = 65535**

**MAX\_RAM = 262144**

**MIN\_CPUS = 1**

**MIN\_PORT = 1024**

**MIN\_RAM = 16**

**REPR\_COLUMNS = ('id', 'hostname', 'port', 'state', 'remote\_ip', 'cpus', 'ram', 'free\_ram')**

**REPR\_CONVERT\_COLUMN = {'remote\_ip': <function repr\_ip at 0x7f3d01b430d0>}**

**STATE\_DEFAULT = 'online'**

**STATE\_ENUM = MappedEnum(ONLINE='online', DISABLED='disabled', RUNNING='running', OFFLINE='offline')**



**URL\_TEMPLATE** = 'http://{host}:{port}/api/v1'

**api\_url** ()

Returns the base url which should be used to access the api of this specific agent.

**Raises ValueError** Raised if this function is called while the agent's *use\_address* column is set to *PASSIVE*

**cpu\_allocation**

The total amount of cpu space an agent is allowed to process work in. A value of 1.0 would mean an agent can handle as much work as the system could handle given the requirements of a task. For example if an agent has 8 cpus, *cpu\_allocation* is .5, and a task requires 4 cpus then only that task will run on the system.

**cpu\_name**

The make and model of CPUs in this agents

**cpus**

The number of logical CPU cores installed on the agent

**disks**

The known disks available to this agent

**failed\_tasks**

The tasks this agents failed to execute

**free\_ram**

The amount of ram which was last considered free

**get\_supported\_types** ()

**gpus**

The graphics cards that are installed in this agent

**hostname**

The hostname we should use to talk to this host. Preferably this value will be the fully qualified name instead of the base hostname alone.

**id**

Provides an id for the current row. This value should never be directly relied upon and it's intended for use by relationships.

**is\_disabled** ()

**is\_offline** ()

**last\_heard\_from**

Time we last had contact with this agent

**last\_polled**

Time we last tried to contact the agent

**last\_success\_on**

The last time this agent has set a task to *done*

**mac\_addresses**

The MAC addresses this agent has

**notes**

Free form notes about this agent

**os\_class**

The type of operating system running on the agent; 'linux', 'windows', or 'mac'.

**os\_fullname**

The full human-readable name of the agent's OS, as returned by `platform.platform()`

**port**

The port the agent is currently running on

**ram**

The amount of ram installed on the agent in megabytes

**ram\_allocation**

The amount of ram the agent is allowed to allocate towards work. A value of 1.0 would mean to let the agent use all of the memory installed on the system when assigning work.

**remote\_ip**

the remote address which came in with the request

**restart\_requested**

If True, the agent will be restarted

**satisfies\_job\_requirements** (*job*)

**satisfies\_jobtype\_requirements** (*jobtype\_version*)

**software\_versions**

software this agent has installed or is configured for

**state**

Stores the current state of the host. This value can be changed either by a master telling the host to do something with a task or from the host via REST api.

**tags**

Tags associated with this agent

**task\_logs**

**tasks**

Relationship between an *Agent* and any `pyfarm.models.Task` objects

**time\_offset**

The offset in seconds the agent is from an official time server

**upgrade\_to**

The version this agent should upgrade to.

**use\_address**

The address we should use when communicating with the agent

**classmethod validate\_hostname** (*key, value*)

Ensures that the hostname provided by *value* matches a regular expression that expresses what a valid hostname is.

**validate\_hostname\_column** (*key, value*)

Validates the hostname column

**classmethod validate\_ipv4\_address** (*\_, value*)

Ensures the ip address is valid. This checks to ensure that the value provided is:

- not a hostmask
- not link local (**RFC 3927**)
- not used for multicast (**RFC 1112**)
- not a netmask (**RFC 4632**)

- not reserved (**RFC 6052**)
- a private address (**RFC 1918**)

**validate\_numeric\_column** (*key, value*)

Validates several numerical columns. Columns such as ram, cpus and port a are validated with this method.

**validate\_remote\_ip** (*key, value*)

Validates the remote\_ip column

**classmethod validate\_resource** (*key, value*)

Ensure the `value` provided for `key` is within an expected range. This classmethod retrieves the min and max values from the `Agent` class directory using:

```
>>> min_value = getattr(Agent, "MIN_%s" % key.upper())
>>> max_value = getattr(Agent, "MAX_%s" % key.upper())
```

**version**

The pyfarm version number this agent is running.

## 4.2.2 pyfarm.models.disk module

### Disk Model

Model describing a given disk, with size and free space.

**class** `pyfarm.models.disk.AgentDisk` (*\*\*kwargs*)

Bases: `flask_sqlalchemy.Model`, `pyfarm.models.core.mixins.UtilityMixins`,  
`pyfarm.models.core.mixins.ReprMixin`

Stores information about a single disk belonging to an agent, including usage information.

**agent**

**agent\_id**

**free**

Available space on the disk in bytes.

**id**

Provides an id for the current row. This value should never be directly relied upon and it's intended for use by relationships.

**mountpoint**

The mountpoint of this disk on the agent (Drive letter for Windows agents)

**size**

The total capacity of this disk in bytes

## 4.2.3 pyfarm.models.gpu module

### GPU Model

Model describing a given make and model of graphics card. Every agent can have zero or more GPUs associated with it.

**class** `pyfarm.models.gpu.GPU` (*\*\*kwargs*)

Bases: `flask_sqlalchemy.Model`, `pyfarm.models.core.mixins.UtilityMixins`,  
`pyfarm.models.core.mixins.ReprMixin`

**agents**

**fullname**

The full name of this graphics card model

**id**

Provides an id for the current row. This value should never be directly relied upon and it's intended for use by relationships.

## 4.2.4 pyfarm.models.job module

### Job Models

Models and interface classes related to jobs.

**class** `pyfarm.models.job.Job` (*\*\*kwargs*)

Bases: `flask_sqlalchemy.Model`, `pyfarm.models.core.mixins.ValidatePriorityMixin`,  
`pyfarm.models.core.mixins.ValidateWorkStateMixin`, `pyfarm.models.core.mixins.WorkStateCh`  
`pyfarm.models.core.mixins.ReprMixin`, `pyfarm.models.core.mixins.UtilityMixins`

Defines the attributes and environment for a job. Individual commands are kept track of by Task

**REPR\_COLUMNS** = ('id', 'state', 'project')

**REPR\_CONVERT\_COLUMN** = {'state': <built-in function repr>}

**STATE\_ENUM** = ['running', 'paused', 'failed', 'done', None]

**alter\_frame\_range** (*start, end, by*)

**autodelete\_time**

If not None, this job will be automatically deleted this number of seconds after it finishes.

**batch**

Number of tasks to run on a single agent at once. Depending on the capabilities of the software being run this will either cause a single process to execute on the agent or multiple processes one after the other.

**by**

The number of frames to count by between *start* and *end*. This column may also sometimes be referred to as 'step' by other software.

**can\_use\_more\_agents** ()

**children**

**clear\_assigned\_counts** ()

**completion\_notify\_sent**

Whether or not the finish notification mail has already been sent out.

**cpus**

Number of cpus or threads each task should consume oneach agent. Depending on the job type being executed this may result in additional cpu consumption, longer wait times in the queue (2 cpus means 2 'fewer' cpus on an agent), or all of the above... csv-table:: **Special Values** :header: Value, Result :widths: 10, 50 0, minimum number of cpu resources not required -1, agent cpu is exclusive for a task from this job

**data**

Json blob containing additional data for a job .. note:: Changes made directly to this object are **not** applied to the session.

**environ**

Dictionary containing information about the environment in which the job will execute. .. note:: Changes made directly to this object are **not** applied to the session.

**get\_batch** (*agent*)**group**

The job group this job belongs to

**hidden**

If True, keep the job hidden from the queue and web ui. This is typically set to True if you either want to save a job for later viewing or if the jobs data is being populated in a deferred manner.

**id**

Provides an id for the current row. This value should never be directly relied upon and it's intended for use by relationships.

**job\_group\_id**

The foreign key which stores: *class:JobGroup.id*

**job\_queue\_id**

The foreign key which stores *JobQueue.id*

**jobtype\_version****jobtype\_version\_id**

The foreign key which stores *JobTypeVersion.id*

**maximum\_agents**

The scheduler will never assign more than this number of agents to this job.

**minimum\_agents**

The scheduler will try to assign at least this number of agents to this job as long as it can use them, before any other considerations.

**notes**

Notes that are provided on submission or added after the fact. This column is only provided for human consumption, is not scanned, indexed, or used when searching

**notified\_users****num\_assigned\_agents** ()**num\_tiles**

How many regions to split frames into for rendering.

**output\_link**

An optional link to a URI where this job's output can be viewed.

**parents****paused** ()**priority**

The priority of the job relative to others in the queue. This is not the same as task priority.

**configured by:** *job.priority*

**queue**

The queue for this job

**ram**

Amount of ram a task from this job will require to be free in order to run. A task exceeding this value will

not result in any special behavior... csv-table:: **Special Values** :header: Value, Result :widths: 10, 500, minimum amount of free ram not required-1, agent ram is exclusive for a task from this job

**ram\_max**

Maximum amount of ram a task is allowed to consume on an agent... warning:: If set, the task will be **terminated** if the ram in use by the process exceeds this value.

**ram\_warning**

Amount of ram used by a task before a warning raised. A task exceeding this value will not cause any work stopping behavior.

**requeue**

Number of times to requeue failed tasks .. csv-table:: **Special Values** :header: Value, Result :widths: 10, 50 0, never requeue failed tasks -1, requeue failed tasks indefinitely

**rerun ()**

Makes this job rerun all its task. Tasks that are currently running are left untouched.

**rerun\_failed ()**

Makes this job rerun all its failed tasks. Tasks that are done or are currently running are left untouched

**software\_requirements**

**state**

The state of the job with a value provided by `WorkState`

**tag\_requirements**

**tags**

Relationship between this job and `Tag` objects

**tasks**

**tasks\_done**

Relationship between this job and any `Task` objects which are done.

**tasks\_failed**

Relationship between this job and any `Task` objects which have failed.

**tasks\_queued**

Relationship between this job and any `Task` objects which are queued.

**tasks\_running**

Relationship between this job and any `Task` objects which are running.

**time\_finished**

Time the job was finished. This will be set when the last task finishes and reset if a job is requeued.

**time\_started**

The time this job was started. By default this value is set when `state` is changed to an appropriate value or when a job is requeued.

**time\_submitted**

The time the job was submitted. By default this defaults to using `datetime.datetime.utcnow()` as the source of submission time. This value will not be set more than once and will not change even after a job is requeued.

**title**

The title of this job

**to\_be\_deleted**

If true, the master will stop all running tasks for this job and then delete it.

**update\_state ()**

**user**  
The owner of this job

**user\_id**  
The id of the user who owns this job

**validate\_progress** (*key, value*)

**validate\_resource** (*key, value*)  
Validation that ensures that the value provided for either *ram* or *cpus* is a valid value with a given range

**weight**  
The weight of this job. The scheduler will distribute available agents between jobs and job queues in the same queue in proportion to their weights.

## 4.2.5 pyfarm.models.jobgroup module

### Job Group Model

Model for job groups

```
class pyfarm.models.jobgroup.JobGroup (**kwargs)
    Bases: flask_sqlalchemy.Model, pyfarm.models.core.mixins.UtilityMixins
```

Used to group jobs together for better presentation in the UI

**id**  
Provides an id for the current row. This value should never be directly relied upon and it's intended for use by relationships.

**jobs**

**main\_jobtype**  
The jobtype of the main job in this group

**main\_jobtype\_id**  
ID of the jobtype of the main job in this group. Purely for display and filtering.

**title**  
The title of the job group's name

**user**  
The user who owns these jobs

**user\_id**  
The id of the user who owns these jobs

## 4.2.6 pyfarm.models.jobqueue module

### Job Queue Model

Model for job queues

```
class pyfarm.models.jobqueue.JobQueue (**kwargs)
    Bases: flask_sqlalchemy.Model, pyfarm.models.core.mixins.UtilityMixins,
    pyfarm.models.core.mixins.ReprMixin
```

Stores information about a job queue. Used for flexible, configurable distribution of computing capacity to jobs.

**REPR\_COLUMNS** = ('id', 'name')

**child\_jobs** (*filters*)

**child\_queues\_sorted** ()

Return child queues sorted by number of currently assigned agents with priority as a secondary sort key.

**children**

**clear\_assigned\_counts** ()

**fullpath**

The path of this jobqueue. This column is a database denormalization. It is technically redundant, but faster to access than recursively querying all parent queues. If set to NULL, the path must be computed by recursively querying the parent queues.

**get\_job\_for\_agent** (*agent, unwanted\_job\_ids=None*)

**id**

Provides an id for the current row. This value should never be directly relied upon and it's intended for use by relationships.

**jobs**

**maximum\_agents**

The scheduler will never assign more than this number of agents to jobs in or below this queue.

**minimum\_agents**

The scheduler will try to assign at least this number of agents to jobs in or below this queue as long as it can use them, before any other considerations.

**name**

The name of the job queue

**num\_assigned\_agents** ()

**parent**

Relationship between this queue its parent

**parent\_jobqueue\_id**

The parent queue of this queue. If NULL, this is a top level queue.

**path** ()

**priority**

The priority of this job queue. The scheduler will not assign any nodes to other job queues or jobs with the same parent and a lower priority as long as this one can still use nodes. The `minimum_agents` column takes precedence over this.

**static top\_level\_unique\_check** (*mapper, connection, target*)

**weight**

The weight of this job queue. The scheduler will distribute available agents between jobs and job queues in the same queue in proportion to their weights.

## 4.2.7 pyfarm.models.jobtype module

### Job Type Models

Models and objects dedicated to handling information which is specific to an individual job. See [pyfarm.models.job](#) for more the more general implementation.



```

class pyfarm.models.jobtype.JobType (**kwargs)
    Bases: flask_sqlalchemy.Model, pyfarm.models.core.mixins.UtilityMixins,
           pyfarm.models.core.mixins.ReprMixin

    Stores the unique information necessary to execute a task

    REPR_COLUMNS = ('id', 'name')

    description
        Human readable description of the job type. This field is not required and is not directly relied upon
        anywhere.

    fail_body
        The email body to use for notifications in in case of success. Some substitutions, for example for the job
        title, are available.

    fail_subject
        The subject line to use for notifications in case of failure. Some substitutions, for example for the job title,
        are available.

    id
        Provides an id for the current row. This value should never be directly relied upon and it's intended for use
        by relationships.

    jobgroups

    name
        The name of the job type. This can be either a human readable name or the name of the job type class
        itself.

    success_body
        The email body to use for notifications in in case of success. Some substitutions, for example for the job
        title, are available.

    success_subject
        The subject line to use for notifications in case of success. Some substitutions, for example for the job
        title, are available.

    validate_name (key, value)

    versions

```

## 4.2.8 pyfarm.models.pathmap module

### Path Map Model

Model for path maps, allowing for OS-dependent mapping of path prefixes to other path prefixes.

```

class pyfarm.models.pathmap.PathMap (**kwargs)
    Bases: flask_sqlalchemy.Model, pyfarm.models.core.mixins.ReprMixin,
           pyfarm.models.core.mixins.UtilityMixins

    Defines a table which is used for cross-platform file path mappings.

    id
        Provides an id for the current row. This value should never be directly relied upon and it's intended for use
        by relationships.

    path_linux
        The path on linux platforms

```

**path\_osx**

The path on Mac OS X platforms

**path\_windows**

The path on Windows platforms

**tag**

Relationship attribute for the tag this path map applies to.

**tag\_id**

The tag an agent needs to have for this path map to apply to it. If this is NULL, this path map applies to all agents, but is overridden by applying path maps that do specify a tag.

## 4.2.9 pyfarm.models.software module

### Software Models

Table of software items. Agents can reference this table to show that they provide a given software. Jobs or jobtypes can depend on a software via the SoftwareRequirement table

**class** `pyfarm.models.software.Software` (\*\*kwargs)

Bases: `flask_sqlalchemy.Model`, `pyfarm.models.core.mixins.UtilityMixins`

Model to represent a versioned piece of software that can be present on an agent and may be depended on by a job and/or jobtype through the appropriate SoftwareRequirement table

**id**

Provides an id for the current row. This value should never be directly relied upon and it's intended for use by relationships.

**software**

The name of the software

**versions**

All known versions of this software

## 4.2.10 pyfarm.models.tag module

### Tag Model

Table with tags for both jobs and agents

**class** `pyfarm.models.tag.Tag` (\*\*kwargs)

Bases: `flask_sqlalchemy.Model`, `pyfarm.models.core.mixins.UtilityMixins`

Model which provides tagging for *Job* and class: *Agent* objects

**agents**

**id**

Provides an id for the current row. This value should never be directly relied upon and it's intended for use by relationships.

**jobs**

**tag**

The actual value of the tag

## 4.2.11 pyfarm.models.task module

### Task Models

Models and interface classes related to tasks

**class** `pyfarm.models.task.Task` (*\*\*kwargs*)

Bases: `flask_sqlalchemy.Model`, `pyfarm.models.core.mixins.ValidatePriorityMixin`,  
`pyfarm.models.core.mixins.ValidateWorkStateMixin`, `pyfarm.models.core.mixins.UtilityMixin`,  
`pyfarm.models.core.mixins.ReprMixin`

Defines a task which a child of a `Job`. This table represents rows which contain the individual work unit(s) for a job.

**REPR\_COLUMNS** = ('id', 'state', 'frame', 'project')

**REPR\_CONVERT\_COLUMN** = {'state': `functools.partial(<function repr_enum at 0x7f3d01b43158>, enum=['running', 'pau`

**STATE\_DEFAULT** = `None`

**STATE\_ENUM** = ['running', 'paused', 'failed', 'done', `None`]

**agent**

**agent\_id**

Foreign key which stores `Job.id`

**attempts**

The number of attempts which have been made on this task. This value is auto incremented when `state` changes to a value synonymous with a running state.

**static clear\_error\_state** (*target, new\_value, old\_value, initiator*)

Sets `last_error` column to `None` if the task's state is 'done'

**failed()**

**failed\_in\_agents**

**failures**

The number of times this task has failed. This value is auto incremented when `state` changes to a value synonymous with a failed state.

**frame**

The frame this `Task` will be executing.

**hidden**

When `True` this hides the task from queue and web ui

**id**

Provides an id for the current row. This value should never be directly relied upon and it's intended for use by relationships.

**static increment\_attempts** (*target, new\_value, old\_value, initiator*)

**job**

relationship attribute which retrieves the associated job for this task

**job\_id**

Foreign key which stores `Job.id`

**last\_error**

This column may be set when an error is present. The agent typically sets this column when the job type either can't or won't run a given task. This column will be cleared whenever the task's state is returned to a non-error state.

`static log_assign_change (target, new_value, old_value, initiator)`

`log_associations`

`priority`

The priority of the job relative to others in the queue. This is not the same as task priority.

**configured by:** `job.priority`

`progress`

The progress for this task, as a value between 0.0 and 1.0. Used purely for display purposes.

`static reset_agent_if_failed_and_retry (target, new_value, old_value, initiator)`

`static reset_finished_time (target, new_value, old_value, initiator)`

`running ()`

`sent_to_agent`

Whether this task was already sent to the assigned agent

`static set_progress_on_success (target, new_value, old_value, initiator)`

`static set_times (target, new_value, old_value, initiator)`

update the datetime objects depending on the new value

`state`

The state of the job with a value provided by `WorkState`

`tile`

When using tiled rendering, the number of the tile this task refers to. The jobtype will have to translate that into an actual image region. This will be NULL if the job doesn't use tiled rendering.

`time_finished`

Time the job was finished. This will be set when the last task finishes and reset if a job is requested.

`time_started`

The time this job was started. By default this value is set when `state` is changed to an appropriate value or when a job is requested.

`time_submitted`

The time the job was submitted. By default this defaults to using `datetime.datetime.utcnow()` as the source of submission time. This value will not be set more than once and will not change even after a job is requested.

`static update_agent_on_success (target, new_value, old_value, initiator)`

`static update_failures (target, new_value, old_value, initiator)`

## 4.2.12 pyfarm.models.tasklog module

### Task Log Models

Model describing a log file for a task or batch of tasks.

A task can be associated with more than one log file, for example because it needed to be retried and there are logs for every attempt or because the job type used uses more than one process to execute a batch. A log file can belong to more than one task if tasks have been batched together for execution.

```
class pyfarm.models.tasklog.TaskLog (**kwargs)
```

```
    Bases: flask_sqlalchemy.Model, pyfarm.models.core.mixins.UtilityMixins,
           pyfarm.models.core.mixins.ReprMixin
```

Table which represents a single task log entry

**agent**

Relationship between an TaskLog and the :class:`pyfarm.models.Agent` it was created on

**agent\_id**

The agent this log was created on

**created\_on**

The time when this log was created

**id**

Provides an id for the current row. This value should never be directly relied upon and it's intended for use by relationships.

**identifier**

The identifier for this log

**num\_done\_tasks()**

**num\_failed\_tasks()**

**num\_queued\_tasks()**

**num\_running\_tasks()**

**task\_associations**

Relationship between tasks and their logs.

**class** `pyfarm.models.tasklog.TaskTaskLogAssociation(**kwargs)`

Bases: `flask_sqlalchemy.Model`

Stores an association between the task table and a task log

**attempt**

The attempt number for the given task log

**log**

**state**

The state of the work being performed

**task**

**task\_id**

The ID of the job a task log is associated with

**task\_log\_id**

The ID of the task log

## 4.2.13 pyfarm.models.user module

### User and Role Models

Stores users and their roles in the database.

**class** `pyfarm.models.user.User(**kwargs)`

Bases: `flask_sqlalchemy.Model`, `flask_login.UserMixin`,  
`pyfarm.models.core.mixins.ReprMixin`

Stores information about a user including the roles they belong to

**REPR\_COLUMNS** = ('id', 'username')

**active**

Enables or disables a particular user across the entire system

**check\_password** (*password*)

checks the password provided against the stored password

**classmethod create** (*username, password, email=None, roles=None*)

**email**

Contact email for registration and possible notifications

**expiration**

User expiration. If this value is set then the user will no longer be able to access PyFarm past the expiration.

**classmethod get** (*id\_or\_username*)

Get a user model either by id or by the user's username

**get\_auth\_token** ()

**get\_id** ()

**has\_roles** (*allowed=None, required=None*)

checks the provided arguments against the roles assigned

**classmethod hash\_password** (*value*)

**id**

**is\_active** ()

returns true if the user and the roles it belongs to are active

**jobgroups**

**jobs**

**last\_login**

The last date that this user was logged in.

**onetime\_code**

SHA256 one time use code which can be used for unique urls such as for password resets.

**password**

The password used to login

**roles**

**subscribed\_jobs**

**username**

The username used to login.

**class** `pyfarm.models.user.Role` (\*\*kwargs)

Bases: `flask_sqlalchemy.Model`

Stores role information that can be used to give a user access to individual resources.

**active**

Enables or disables a role. Disabling a role will prevent any users of this role from accessing PyFarm

**classmethod create** (*name, description=None*)

Creates a role by the given name or returns an existing role if it already exists.

**description**

Human description of the role.

**expiration**

Role expiration. If this value is set then the role, and anyone assigned to it, will no longer be able to access PyFarm past the expiration.

**id**

**is\_active()**

**name**

The name of the role

**users**

## 4.3 Module contents

Contains all the models used for database communication and object relational management.





---

## pyfarm.scheduler package

---

### 5.1 Submodules

#### 5.1.1 pyfarm.scheduler.celery\_app module

##### Celery Application

Creates the base instance of `Celery` which is used by components of PyFarm's master that require interaction with a task queue. This module also configures Celery's beat scheduler for other tasks such as agent polling and task assignment.

#### 5.1.2 pyfarm.scheduler.statistics\_tasks module

##### Tasks For Statistics

This module contains various celery tasks for gathering runtime statistics about the farm.

#### 5.1.3 pyfarm.scheduler.tasks module

##### Tasks

This module contains various asynchronous tasks to be run by celery.

`pyfarm.scheduler.tasks.send_email(to, message)`

Configures and instance of SMTP and sends a message to the given address.

### 5.2 Module contents

#### 5.2.1 Scheduler

This package contains the components used by PyFarm's master to schedule tasks, assign work to agents, and other periodic tasks that can't be performed inside of the web application.



---

## Indices and tables

---

- `genindex`
- `modindex`
- `search`



**/api**

	GET /api/v1/jobs/schema HTTP/1.1, 46
GET /api/v1/agents/ HTTP/1.1, 20	GET /api/v1/jobtypes/ HTTP/1.1, 47
GET /api/v1/agents/(str:agent_id) HTTP/1.1, 23	GET /api/v1/jobtypes/<str:tagname> HTTP/1.1, 52
GET /api/v1/agents/<str:agent_id>/software/ HTTP/1.1, 25	GET /api/v1/jobtypes/[<str:name> <int:id>]/software/ HTTP/1.1, 50
GET /api/v1/agents/<str:agent_id>/tasks/ HTTP/1.1, 26	GET /api/v1/jobtypes/[<str:name> <int:id>]/software/ HTTP/1.1, 49
GET /api/v1/agents/schema HTTP/1.1, 28	GET /api/v1/jobtypes/[<str:name> <int:id>]/versions/ HTTP/1.1, 51
GET /api/v1/agents/updates/<string:version> HTTP/1.1, 19	GET /api/v1/jobtypes/[<str:name> <int:id>]/versions/ HTTP/1.1, 54
GET /api/v1/jobgroups/ HTTP/1.1, 28	GET /api/v1/jobtypes/[<str:name> <int:id>]/versions/ HTTP/1.1, 47
GET /api/v1/jobgroups/<int:id> HTTP/1.1, 30	GET /api/v1/jobtypes/schema HTTP/1.1, 55
GET /api/v1/jobgroups/<int:id>/jobs HTTP/1.1, 29	GET /api/v1/pathmaps/ HTTP/1.1, 56
GET /api/v1/jobgroups/schema HTTP/1.1, 31	GET /api/v1/pathmaps/<int:pathmap_id> HTTP/1.1, 58
GET /api/v1/jobqueues/ HTTP/1.1, 32	GET /api/v1/pathmaps/schema HTTP/1.1, 59
GET /api/v1/jobqueues/[<str:name> <int:id>] HTTP/1.1, 34	GET /api/v1/software/ HTTP/1.1, 62
GET /api/v1/jobqueues/schema HTTP/1.1, 35	GET /api/v1/software/<str:softwarename> HTTP/1.1, 60
GET /api/v1/jobs/ HTTP/1.1, 35	GET /api/v1/software/<str:softwarename>/versions/ HTTP/1.1, 64
GET /api/v1/jobs/<int:job_id>/tasks/<int:task_id>/failed_on_agents/ HTTP/1.1, 45	GET /api/v1/software/<str:softwarename>/versions/<int:software_id>/ HTTP/1.1, 62
GET /api/v1/jobs/<job_id>/tasks/<task_id>/attempts/<attempt>/logs/ HTTP/1.1, 71	GET /api/v1/software/[<str:software_name> <int:software_id>]/versions/<int:software_id>/ HTTP/1.1, 63
GET /api/v1/jobs/<job_id>/tasks/<task_id>/attempts/<attempt>/logs/<log_identifier> HTTP/1.1, 72	GET /api/v1/software/schema HTTP/1.1, 65
GET /api/v1/jobs/<job_id>/tasks/<task_id>/attempts/<attempt>/logs/<log_identifier>/logfile HTTP/1.1, 73	GET /api/v1/tags/<str:tagname> HTTP/1.1, 68
GET /api/v1/jobs/[<str:name> <int:id>] HTTP/1.1, 42	GET /api/v1/tags/<str:tagname>/agents/ HTTP/1.1, 66
GET /api/v1/jobs/[<str:name> <int:id>]/notified_users/ HTTP/1.1, 37	GET /api/v1/tags/schema/ HTTP/1.1, 71
GET /api/v1/jobs/[<str:name> <int:id>]/tasks/ HTTP/1.1, 40	POST /api/v1/agents/ HTTP/1.1, 21
GET /api/v1/jobs/[<str:name> <int:id>]/tasks/<int:task_id> HTTP/1.1, 39	POST /api/v1/agents/(str:agent_id) HTTP/1.1, 24

```

DELETE /api/v1/jobs/[[<str:name>|<int:id>]
HTTP/1.1, 41
POST /api/v1/agents/<str:agent_id>/software/ HTTP/1.1, 25
DELETE /api/v1/jobs/[[<str:name>|<int:id>]/notified_
HTTP/1.1, 38
POST /api/v1/agents/<str:agent_id>/tasks/ HTTP/1.1, 27
DELETE /api/v1/jobtypes/[[<str:name>|<int:id>]
HTTP/1.1, 52
POST /api/v1/jobgroups/ HTTP/1.1, 29
DELETE /api/v1/jobtypes/[[<str:name>|<int:id>]/softw
HTTP/1.1, 31
DELETE /api/v1/jobtypes/[[<str:name>|<int:id>]/vers
HTTP/1.1, 33
POST /api/v1/jobqueues/ HTTP/1.1, 33
DELETE /api/v1/pathmaps/<int:pathmap_id>
HTTP/1.1, 34
DELETE /api/v1/pathmaps/<int:pathmap_id>/agents/
HTTP/1.1, 36
DELETE /api/v1/software/<str:softwarename>
HTTP/1.1, 45
DELETE /api/v1/software/<str:softwarename>/version
HTTP/1.1, 49
POST /api/v1/jobs/<job_id>/tasks/<task_id>/attempts/<attempt>/logs/
HTTP/1.1, 72
DELETE /api/v1/software/<str:softwarename>/version
HTTP/1.1, 73
POST /api/v1/jobs/<job_id>/tasks/<task_id>/attempts/<attempt>/logs/<log_identifier>
HTTP/1.1, 73
DELETE /api/v1/tags/<str:tagname>
HTTP/1.1, 43
POST /api/v1/jobs/[[<str:name>|<int:id>]
HTTP/1.1, 67
POST /api/v1/jobs/[[<str:name>|<int:id>]/notified_users/
HTTP/1.1, 38
POST /api/v1/jobs/[[<str:name>|<int:id>]/tasks/<int:task_id>
HTTP/1.1, 40
POST /api/v1/jobtypes/ HTTP/1.1, 48
POST /api/v1/jobtypes/[[<str:name>|<int:id>]/software_requirements/
HTTP/1.1, 51
POST /api/v1/pathmaps/ HTTP/1.1, 57
POST /api/v1/pathmaps/<int:pathmap_id>
HTTP/1.1, 58
POST /api/v1/software/ HTTP/1.1, 63
POST /api/v1/software/versions/
HTTP/1.1, 65
POST /api/v1/tags/ HTTP/1.1, 70
POST /api/v1/tags/<str:tagname>/agents/
HTTP/1.1, 66
PUT /api/v1/agents/updates/<string:version>
HTTP/1.1, 19
PUT /api/v1/jobs/<job_id>/tasks/<task_id>/attempts/<attempt>/logs/<log_identifier>/logfile
HTTP/1.1, 74
PUT /api/v1/jobtypes/[[<str:name>|<int:id>]
HTTP/1.1, 53
PUT /api/v1/software/<str:softwarename>
HTTP/1.1, 60
PUT /api/v1/tags/<str:tagname>
HTTP/1.1, 68
DELETE /api/v1/agents/(uuid:agent_id)
HTTP/1.1, 23
DELETE /api/v1/agents/<str:agent_id>/software/<str:software>/versions/<str:version>
HTTP/1.1, 24
DELETE /api/v1/jobgroup/<int:id>, 30
DELETE /api/v1/jobqueue/[[<str:name>|<int:id>],
33
DELETE /api/v1/jobs/<int:id>/tasks/<int:task_id>/failed_on_agents/<str:agent_id>
HTTP/1.1, 44

```

**p**

- pyfarm.master, 84
- pyfarm.master.api, 74
  - pyfarm.master.api.agent\_updates, 19
  - pyfarm.master.api.agents, 20
  - pyfarm.master.api.jobgroups, 28
  - pyfarm.master.api.jobqueues, 32
  - pyfarm.master.api.jobs, 35
  - pyfarm.master.api.jobtypes, 47
  - pyfarm.master.api.pathmaps, 56
  - pyfarm.master.api.software, 59
  - pyfarm.master.api.tags, 66
  - pyfarm.master.api.tasklogs, 71
- pyfarm.master.application, 78
- pyfarm.master.config, 79
- pyfarm.master.entrypoints, 79
- pyfarm.master.index, 80
- pyfarm.master.initial, 80
- pyfarm.master.testutil, 80
- pyfarm.master.user\_interface, 78
  - pyfarm.master.user\_interface.agents, 75
  - pyfarm.master.user\_interface.jobgroups, 76
  - pyfarm.master.user\_interface.jobqueues, 76
  - pyfarm.master.user\_interface.jobs, 76
  - pyfarm.master.user\_interface.jobtypes, 77
  - pyfarm.master.user\_interface.software, 77
  - pyfarm.master.user\_interface.software\_version, 77
  - pyfarm.master.user\_interface.statistics, 75
    - pyfarm.master.user\_interface.statistics.agent\_counts, 75
    - pyfarm.master.user\_interface.statistics.index, 75
    - pyfarm.master.user\_interface.statistics.task\_events, 75
- pyfarm.master.utility, 82
- pyfarm.models, 107
  - pyfarm.models.agent, 92
  - pyfarm.models.core, 90
    - pyfarm.models.core.functions, 85
    - pyfarm.models.core.mixins, 86
    - pyfarm.models.core.types, 88
  - pyfarm.models.disk, 95
  - pyfarm.models.gpu, 95
  - pyfarm.models.job, 96
  - pyfarm.models.jobgroup, 99
  - pyfarm.models.jobqueue, 99
  - pyfarm.models.jobtype, 100
  - pyfarm.models.pathmap, 101
  - pyfarm.models.software, 102
  - pyfarm.models.statistics, 92
    - pyfarm.models.statistics.agent\_count, 90
    - pyfarm.models.statistics.task\_count, 91
    - pyfarm.models.statistics.task\_event\_count, 91
  - pyfarm.models.tag, 102
  - pyfarm.models.task, 103
  - pyfarm.models.tasklog, 104
  - pyfarm.models.user, 105
- pyfarm.scheduler, 109
  - pyfarm.scheduler.celery\_app, 109
  - pyfarm.scheduler.statistics\_tasks, 109
  - pyfarm.scheduler.tasks, 109





## A

- active (pyfarm.models.user.Role attribute), 106
- active (pyfarm.models.user.User attribute), 106
- add\_jobtype\_software\_requirement() (in module pyfarm.master.user\_interface.jobtypes), 77
- add\_notified\_user\_to\_job() (in module pyfarm.master.user\_interface.jobs), 76
- add\_sample() (pyfarm.master.user\_interface.statistics.task\_events.TotalsAverage method), 75
- add\_software() (in module pyfarm.master.user\_interface.software), 77
- add\_software\_version() (in module pyfarm.master.user\_interface.software), 77
- add\_tag\_on\_jobs() (in module pyfarm.master.user\_interface.jobs), 76
- add\_tag\_requirement\_on\_jobs() (in module pyfarm.master.user\_interface.jobs), 76
- Agent (class in pyfarm.models.agent), 92
- agent (pyfarm.models.disk.AgentDisk attribute), 95
- agent (pyfarm.models.task.Task attribute), 103
- agent (pyfarm.models.tasklog.TaskLog attribute), 105
- agent\_add\_software() (in module pyfarm.master.user\_interface.agents), 75
- agent\_counts() (in module pyfarm.master.user\_interface.statistics.agent\_counts), 75
- agent\_delete\_software() (in module pyfarm.master.user\_interface.agents), 75
- agent\_id (pyfarm.models.disk.AgentDisk attribute), 95
- agent\_id (pyfarm.models.task.Task attribute), 103
- agent\_id (pyfarm.models.tasklog.TaskLog attribute), 105
- AgentCount (class in pyfarm.models.statistics.agent\_count), 90
- AgentDisk (class in pyfarm.models.disk), 95
- AgentIndexAPI (class in pyfarm.master.api.agents), 20
- agents (pyfarm.models.gpu.GPU attribute), 95
- agents (pyfarm.models.tag.Tag attribute), 102
- agents() (in module pyfarm.master.user\_interface.agents), 75
- AgentsInTagIndexAPI (class in pyfarm.master.api.tags), 66
- AgentStateEnum (class in pyfarm.models.core.types), 88
- AgentUpdatesAPI (class in pyfarm.master.api.agent\_updates), 19
- alter\_autodeletion\_for\_job() (in module pyfarm.master.user\_interface.jobs), 76
- alter\_frame\_range() (pyfarm.models.job.Job method), 96
- alter\_frames\_in\_single\_job() (in module pyfarm.master.user\_interface.jobs), 76
- alter\_scheduling\_parameters\_for\_job() (in module pyfarm.master.user\_interface.jobs), 76
- api\_url() (pyfarm.models.agent.Agent method), 93
- assert\_accepted() (pyfarm.master.testutil.BaseTestCase method), 80
- assert\_bad\_request() (pyfarm.master.testutil.BaseTestCase method), 80
- assert\_conflict() (pyfarm.master.testutil.BaseTestCase method), 80
- assert\_contents\_equal() (pyfarm.master.testutil.BaseTestCase method), 80
- assert\_created() (pyfarm.master.testutil.BaseTestCase method), 81
- assert\_forbidden() (pyfarm.master.testutil.BaseTestCase method), 81
- assert\_internal\_server\_error() (pyfarm.master.testutil.BaseTestCase method), 81
- assert\_method\_not\_allowed() (pyfarm.master.testutil.BaseTestCase method), 81
- assert\_mimetypes() (in module pyfarm.master.utility), 82
- assert\_no\_content() (pyfarm.master.testutil.BaseTestCase method), 81
- assert\_not\_acceptable() (pyfarm.master.testutil.BaseTestCase method), 81
- assert\_not\_found() (pyfarm.master.testutil.BaseTestCase method), 81

assert\_ok() (pyfarm.master.testutil.BaseTestCase method), 81  
 assert\_status() (pyfarm.master.testutil.BaseTestCase method), 81  
 assert\_temporary\_redirect() (pyfarm.master.testutil.BaseTestCase method), 81  
 assert\_unauthorized() (pyfarm.master.testutil.BaseTestCase method), 81  
 assert\_unsupported\_media\_type() (pyfarm.master.testutil.BaseTestCase method), 81  
 attempt (pyfarm.models.tasklog.TaskTaskLogAssociation attribute), 105  
 attempts (pyfarm.models.task.Task attribute), 103  
 autodelete\_time (pyfarm.models.job.Job attribute), 96  
 autoincrementing (pyfarm.models.core.mixins.ModelTypes attribute), 86  
 avg\_done() (pyfarm.master.user\_interface.statistics.task\_events.TotalAverage method), 75  
 avg\_failed() (pyfarm.master.user\_interface.statistics.task\_events.TotalAverage method), 75  
 avg\_queued() (pyfarm.master.user\_interface.statistics.task\_events.TotalAverage method), 75  
 avg\_running() (pyfarm.master.user\_interface.statistics.task\_events.TotalAverage method), 75  
 children (pyfarm.models.jobqueue.JobQueue attribute), 100  
 clear\_assigned\_counts() (pyfarm.models.job.Job method), 96  
 clear\_assigned\_counts() (pyfarm.models.jobqueue.JobQueue method), 100  
 clear\_error\_state() (pyfarm.models.task.Task static method), 103  
 columns (pyfarm.models.core.mixins.ModelTypes attribute), 86  
 completion\_notify\_sent (pyfarm.models.job.Job attribute), 96  
 Configuration (class in pyfarm.master.config), 79  
 counted\_time (pyfarm.models.statistics.agent\_count.AgentCount attribute), 90  
 counted\_time (pyfarm.models.statistics.task\_count.TaskCount attribute), 91  
 cpu\_allocation (pyfarm.models.agent.Agent attribute), 93  
 cpus (pyfarm.models.agent.Agent attribute), 93  
 create() (pyfarm.models.agent.Agent attribute), 93  
 create() (pyfarm.models.job.Job attribute), 96  
 create() (pyfarm.models.user.Role class method), 106  
 create() (pyfarm.models.user.User class method), 106  
 create\_app() (in module pyfarm.master.entrypoints), 79  
 create\_job\_type() (in module pyfarm.master.user\_interface.jobtypes), 77  
 created\_on (pyfarm.models.tasklog.TaskLog attribute), 105

## B

BaseTestCase (class in pyfarm.master.testutil), 80  
 batch (pyfarm.models.job.Job attribute), 96  
 before\_request() (in module pyfarm.master.application), 78  
 build\_environment() (pyfarm.master.testutil.BaseTestCase class method), 81  
 by (pyfarm.models.job.Job attribute), 96

## C

can\_use\_more\_agents() (pyfarm.models.job.Job method), 96  
 check\_password() (pyfarm.models.user.User method), 106  
 check\_software\_in\_single\_agent() (in module pyfarm.master.user\_interface.agents), 75  
 checkInteger() (pyfarm.models.core.types.IPv4Address method), 88  
 child\_jobs() (pyfarm.models.jobqueue.JobQueue method), 100  
 child\_queues\_sorted() (pyfarm.models.jobqueue.JobQueue method), 100  
 children (pyfarm.models.job.Job attribute), 96

## D

data (pyfarm.models.job.Job attribute), 96  
 default() (pyfarm.master.utility.JSONEncoder method), 82  
 default\_json\_encoder() (in module pyfarm.master.utility), 82  
 delete() (pyfarm.master.api.agents.SingleAgentAPI method), 23  
 delete() (pyfarm.master.api.agents.SingleSoftwareInAgentAPI method), 24  
 delete() (pyfarm.master.api.jobgroups.SingleJobGroupAPI method), 30  
 delete() (pyfarm.master.api.jobqueues.SingleJobQueueAPI method), 33  
 delete() (pyfarm.master.api.jobs.JobSingleNotifiedUserAPI method), 38  
 delete() (pyfarm.master.api.jobs.SingleJobAPI method), 41  
 delete() (pyfarm.master.api.jobs.SingleTaskOnAgentFailureAPI method), 44  
 delete() (pyfarm.master.api.jobtypes.JobTypeSoftwareRequirementAPI method), 49  
 delete() (pyfarm.master.api.jobtypes.SingleJobTypeAPI method), 52

- delete() (pyfarm.master.api.jobtypes.VersionedJobTypeAPI method), 54
  - delete() (pyfarm.master.api.pathmaps.SinglePathMapAPI method), 57
  - delete() (pyfarm.master.api.software.SingleSoftwareAPI method), 59
  - delete() (pyfarm.master.api.software.SingleSoftwareVersionAPI method), 61
  - delete() (pyfarm.master.api.tags.SingleTagAPI method), 67
  - delete\_jobqueue() (in module pyfarm.master.user\_interface.jobqueues), 76
  - delete\_multiple\_agents() (in module pyfarm.master.user\_interface.agents), 75
  - delete\_multiple\_jobs() (in module pyfarm.master.user\_interface.jobs), 76
  - delete\_single\_agent() (in module pyfarm.master.user\_interface.agents), 75
  - delete\_single\_job() (in module pyfarm.master.user\_interface.jobs), 76
  - delete\_subqueue() (in module pyfarm.master.user\_interface.jobqueues), 76
  - description (pyfarm.models.jobtype.JobType attribute), 101
  - description (pyfarm.models.user.Role attribute), 106
  - DICT\_CONVERT\_COLUMN (pyfarm.models.core.mixins.UtilityMixins attribute), 87
  - disable\_multiple\_agents() (in module pyfarm.master.user\_interface.agents), 75
  - disable\_single\_agent() (in module pyfarm.master.user\_interface.agents), 75
  - disks (pyfarm.models.agent.Agent attribute), 93
  - dumps() (in module pyfarm.master.utility), 82
  - dumps() (pyfarm.models.core.types.JSONSerializable method), 89
- ## E
- email (pyfarm.master.initial.NewUserForm attribute), 80
  - email (pyfarm.models.user.User attribute), 106
  - enable\_multiple\_agents() (in module pyfarm.master.user\_interface.agents), 75
  - enable\_single\_agent() (in module pyfarm.master.user\_interface.agents), 75
  - enum (pyfarm.models.core.types.AgentStateEnum attribute), 88
  - enum (pyfarm.models.core.types.EnumType attribute), 88
  - enum (pyfarm.models.core.types.OperatingSystemEnum attribute), 90
  - enum (pyfarm.models.core.types.UseAgentAddressEnum attribute), 90
  - enum (pyfarm.models.core.types.WorkStateEnum attribute), 90
  - EnumType (class in pyfarm.models.core.types), 88
- environ (pyfarm.models.job.Job attribute), 96
  - ENVIRONMENT\_OVERRIDES (pyfarm.master.config.Configuration attribute), 79
  - ENVIRONMENT\_SETUP (pyfarm.master.testutil.BaseTestCase attribute), 80
  - error\_handler() (in module pyfarm.master.utility), 82
  - expiration (pyfarm.models.user.Role attribute), 106
  - expiration (pyfarm.models.user.User attribute), 106
  - extract\_version\_dicts() (in module pyfarm.master.api.software), 65
- ## F
- fail\_body (pyfarm.models.jobtype.JobType attribute), 101
  - fail\_missing\_assignments() (in module pyfarm.master.api.agents), 28
  - fail\_subject (pyfarm.models.jobtype.JobType attribute), 101
  - failed() (pyfarm.models.task.Task method), 103
  - failed\_in\_agents (pyfarm.models.task.Task attribute), 103
  - failed\_tasks (pyfarm.models.agent.Agent attribute), 93
  - failures (pyfarm.models.task.Task attribute), 103
  - favicon() (in module pyfarm.master.index), 80
  - frame (pyfarm.models.task.Task attribute), 103
  - free (pyfarm.models.disk.AgentDisk attribute), 95
  - free\_ram (pyfarm.models.agent.Agent attribute), 93
  - fullname (pyfarm.models.gpu.GPU attribute), 96
  - fullpath (pyfarm.models.jobqueue.JobQueue attribute), 100
- ## G
- get() (pyfarm.master.api.agent\_updates.AgentUpdatesAPI method), 19
  - get() (pyfarm.master.api.agents.AgentIndexAPI method), 20
  - get() (pyfarm.master.api.agents.SingleAgentAPI method), 23
  - get() (pyfarm.master.api.agents.SoftwareInAgentIndexAPI method), 25
  - get() (pyfarm.master.api.agents.TasksInAgentAPI method), 26
  - get() (pyfarm.master.api.jobgroups.JobGroupIndexAPI method), 28
  - get() (pyfarm.master.api.jobgroups.JobsInJobGroupIndexAPI method), 29
  - get() (pyfarm.master.api.jobgroups.SingleJobGroupAPI method), 30
  - get() (pyfarm.master.api.jobqueues.JobQueueIndexAPI method), 32
  - get() (pyfarm.master.api.jobqueues.SingleJobQueueAPI method), 33
  - get() (pyfarm.master.api.jobs.JobIndexAPI method), 35

- get() (pyfarm.master.api.jobs.JobNotifiedUsersIndexAPI method), 37
  - get() (pyfarm.master.api.jobs.JobSingleTaskAPI method), 39
  - get() (pyfarm.master.api.jobs.JobTasksIndexAPI method), 40
  - get() (pyfarm.master.api.jobs.SingleJobAPI method), 42
  - get() (pyfarm.master.api.jobs.TaskFailedOnAgentsIndexAPI method), 45
  - get() (pyfarm.master.api.jobtypes.JobTypeCodeAPI method), 47
  - get() (pyfarm.master.api.jobtypes.JobTypeIndexAPI method), 47
  - get() (pyfarm.master.api.jobtypes.JobTypeSoftwareRequirementAPI method), 49
  - get() (pyfarm.master.api.jobtypes.JobTypeSoftwareRequirementIndexAPI method), 50
  - get() (pyfarm.master.api.jobtypes.JobTypeVersionsIndexAPI method), 51
  - get() (pyfarm.master.api.jobtypes.SingleJobTypeAPI method), 52
  - get() (pyfarm.master.api.jobtypes.VersionedJobTypeAPI method), 54
  - get() (pyfarm.master.api.pathmaps.PathMapIndexAPI method), 56
  - get() (pyfarm.master.api.pathmaps.SinglePathMapAPI method), 58
  - get() (pyfarm.master.api.software.SingleSoftwareAPI method), 59
  - get() (pyfarm.master.api.software.SingleSoftwareVersionAPI method), 62
  - get() (pyfarm.master.api.software.SoftwareIndexAPI method), 62
  - get() (pyfarm.master.api.software.SoftwareVersionDiscoveryCodeAPI method), 63
  - get() (pyfarm.master.api.software.SoftwareVersionsIndexAPI method), 64
  - get() (pyfarm.master.api.tags.AgentsInTagIndexAPI method), 66
  - get() (pyfarm.master.api.tags.SingleTagAPI method), 67
  - get() (pyfarm.master.api.tags.TagIndexAPI method), 69
  - get() (pyfarm.master.api.tasklogs.LogsInTaskAttemptsIndexAPI method), 71
  - get() (pyfarm.master.api.tasklogs.SingleLogInTaskAttempt method), 72
  - get() (pyfarm.master.api.tasklogs.TaskLogfileAPI method), 73
  - get() (pyfarm.models.user.User class method), 106
  - get\_api\_blueprint() (in module pyfarm.master.application), 78
  - get\_application() (in module pyfarm.master.application), 78
  - get\_auth\_token() (pyfarm.models.user.User method), 106
  - get\_batch() (pyfarm.models.job.Job method), 97
  - get\_g() (in module pyfarm.master.utility), 83
  - get\_id() (pyfarm.models.user.User method), 106
  - get\_job\_for\_agent() (pyfarm.models.jobqueue.JobQueue method), 100
  - get\_login\_manager() (in module pyfarm.master.application), 78
  - get\_login\_serializer() (in module pyfarm.master.application), 78
  - get\_request\_argument() (in module pyfarm.master.utility), 83
  - get\_sqlalchemy() (in module pyfarm.master.application), 78
  - get\_supported\_types() (pyfarm.models.agent.Agent method), 93
  - getuuid() (in module pyfarm.models.core.functions), 85
  - GPUIndexAPI (pyfarm.models.gpu), 95
  - gpus (pyfarm.models.agent.Agent attribute), 93
  - group (pyfarm.models.job.Job attribute), 97
- ## H
- has\_roles() (pyfarm.models.user.User method), 106
  - hash\_password() (pyfarm.models.user.User class method), 106
  - hidden (pyfarm.models.job.Job attribute), 97
  - hidden (pyfarm.models.task.Task attribute), 103
  - hostname (pyfarm.models.agent.Agent attribute), 93
- ## I
- id (pyfarm.models.agent.Agent attribute), 93
  - id (pyfarm.models.disk.AgentDisk attribute), 95
  - id (pyfarm.models.gpu.GPU attribute), 96
  - id (pyfarm.models.job.Job attribute), 97
  - id (pyfarm.models.jobgroup.JobGroup attribute), 99
  - id (pyfarm.models.jobqueue.JobQueue attribute), 100
  - id (pyfarm.models.jobtype.JobType attribute), 101
  - id (pyfarm.models.pathmap.PathMap attribute), 101
  - id (pyfarm.models.software.Software attribute), 102
  - id (pyfarm.models.statistics.task\_count.TaskCount attribute), 91
  - id (pyfarm.models.statistics.task\_event\_count.TaskEventCount attribute), 91
  - id (pyfarm.models.tag.Tag attribute), 102
  - id (pyfarm.models.task.Task attribute), 103
  - id (pyfarm.models.tasklog.TaskLog attribute), 105
  - id (pyfarm.models.user.Role attribute), 107
  - id (pyfarm.models.user.User attribute), 106
  - id\_column() (in module pyfarm.models.core.types), 90
  - identifier (pyfarm.models.tasklog.TaskLog attribute), 105
  - IDTypeAgent (in module pyfarm.models.core.types), 88
  - impl (pyfarm.models.core.types.EnumType attribute), 88
  - impl (pyfarm.models.core.types.IPv4Address attribute), 88
  - impl (pyfarm.models.core.types.JSONSerializable attribute), 89

- impl (pyfarm.models.core.types.MACAddress attribute), 89
  - impl (pyfarm.models.core.types.UUIDType attribute), 90
  - increment\_attempts() (pyfarm.models.task.Task static method), 103
  - index\_page() (in module pyfarm.master.index), 80
  - inside\_request() (in module pyfarm.master.utility), 83
  - IPAddress (class in pyfarm.models.core.types), 88
  - IPv4Address (class in pyfarm.models.core.types), 88
  - is\_active() (pyfarm.models.user.Role method), 107
  - is\_active() (pyfarm.models.user.User method), 106
  - is\_disabled() (pyfarm.models.agent.Agent method), 93
  - is\_offline() (pyfarm.models.agent.Agent method), 93
  - isuuid() (in module pyfarm.master.utility), 83
- ## J
- Job (class in pyfarm.models.job), 96
  - job (pyfarm.models.task.Task attribute), 103
  - job\_group\_id (pyfarm.models.job.Job attribute), 97
  - job\_id (pyfarm.models.task.Task attribute), 103
  - job\_queue\_id (pyfarm.models.job.Job attribute), 97
  - job\_queue\_id (pyfarm.models.statistics.task\_count.TaskCount attribute), 91
  - job\_queue\_id (pyfarm.models.statistics.task\_event\_count.TaskEventCount attribute), 91
  - JobGroup (class in pyfarm.models.jobgroup), 99
  - JobGroupIndexAPI (class in pyfarm.master.api.jobgroups), 28
  - jobgroups (pyfarm.models.jobtype.JobType attribute), 101
  - jobgroups (pyfarm.models.user.User attribute), 106
  - jobgroups() (in module pyfarm.master.user\_interface.jobgroups), 76
  - JobIndexAPI (class in pyfarm.master.api.jobs), 35
  - JobNotifiedUsersIndexAPI (class in pyfarm.master.api.jobs), 37
  - JobQueue (class in pyfarm.models.jobqueue), 99
  - jobqueue() (in module pyfarm.master.user\_interface.jobqueues), 76
  - jobqueue\_create() (in module pyfarm.master.user\_interface.jobqueues), 76
  - JobQueueIndexAPI (class in pyfarm.master.api.jobqueues), 32
  - jobqueues() (in module pyfarm.master.user\_interface.jobqueues), 76
  - jobs (pyfarm.models.jobgroup.JobGroup attribute), 99
  - jobs (pyfarm.models.jobqueue.JobQueue attribute), 100
  - jobs (pyfarm.models.tag.Tag attribute), 102
  - jobs (pyfarm.models.user.User attribute), 106
  - jobs() (in module pyfarm.master.user\_interface.jobs), 76
  - JobSingleNotifiedUserAPI (class in pyfarm.master.api.jobs), 38
  - JobSingleTaskAPI (class in pyfarm.master.api.jobs), 39
  - JobsInJobGroupIndexAPI (class in pyfarm.master.api.jobgroups), 29
  - JobTasksIndexAPI (class in pyfarm.master.api.jobs), 40
  - JobType (class in pyfarm.models.jobtype), 100
  - jobtype() (in module pyfarm.master.user\_interface.jobtypes), 77
  - jobtype\_version (pyfarm.models.job.Job attribute), 97
  - jobtype\_version\_id (pyfarm.models.job.Job attribute), 97
  - JobTypeCodeAPI (class in pyfarm.master.api.jobtypes), 47
  - JobTypeIndexAPI (class in pyfarm.master.api.jobtypes), 47
  - jobtypes() (in module pyfarm.master.user\_interface.jobtypes), 77
  - JobTypeSoftwareRequirementAPI (class in pyfarm.master.api.jobtypes), 49
  - JobTypeSoftwareRequirementsIndexAPI (class in pyfarm.master.api.jobtypes), 50
  - JobTypeVersionsIndexAPI (class in pyfarm.master.api.jobtypes), 51
  - json (pyfarm.master.testutil.JsonResponseMixin attribute), 81
  - json\_types (pyfarm.models.core.types.EnumType attribute), 88
  - json\_types (pyfarm.models.core.types.IPv4Address attribute), 88
  - json\_types (pyfarm.models.core.types.JSONDict attribute), 88
  - json\_types (pyfarm.models.core.types.JSONList attribute), 89
  - json\_types (pyfarm.models.core.types.MACAddress attribute), 89
  - json\_types (pyfarm.models.core.types.UUIDType attribute), 90
  - JSONDict (class in pyfarm.models.core.types), 88
  - JSONEncoder (class in pyfarm.master.utility), 82
  - jsonify() (in module pyfarm.master.utility), 83
  - JSONList (class in pyfarm.models.core.types), 89
  - JsonResponseMixin (class in pyfarm.master.testutil), 81
  - JSONSerializable (class in pyfarm.models.core.types), 89
- ## L
- last\_error (pyfarm.models.task.Task attribute), 103
  - last\_heard\_from (pyfarm.models.agent.Agent attribute), 93
  - last\_login (pyfarm.models.user.User attribute), 106
  - last\_polled (pyfarm.models.agent.Agent attribute), 93
  - last\_success\_on (pyfarm.models.agent.Agent attribute), 93
  - load\_api() (in module pyfarm.master.entrypoints), 79
  - load\_authentication() (in module pyfarm.master.entrypoints), 79
  - load\_before\_first() (in module pyfarm.master.entrypoints), 79



[load\\_dialect\\_impl\(\)](#) (pyfarm.models.core.types.UUIDType method), 90  
[load\\_error\\_handlers\(\)](#) (in module pyfarm.master.entrypoints), 79  
[load\\_index\(\)](#) (in module pyfarm.master.entrypoints), 79  
[load\\_master\(\)](#) (in module pyfarm.master.entrypoints), 79  
[load\\_setup\(\)](#) (in module pyfarm.master.entrypoints), 79  
[load\\_user\\_interface\(\)](#) (in module pyfarm.master.entrypoints), 79  
[log](#) (pyfarm.models.tasklog.TaskTaskLogAssociation attribute), 105  
[log\\_assign\\_change\(\)](#) (pyfarm.models.task.Task static method), 103  
[log\\_associations](#) (pyfarm.models.task.Task attribute), 104  
[LogsInTaskAttemptsIndexAPI](#) (class in pyfarm.master.api.tasklogs), 71

## M

[mac\\_addresses](#) (pyfarm.models.agent.Agent attribute), 93  
[MACAddress](#) (class in pyfarm.models.core.types), 89  
[main\\_jobtype](#) (pyfarm.models.jobgroup.JobGroup attribute), 99  
[main\\_jobtype\\_id](#) (pyfarm.models.jobgroup.JobGroup attribute), 99  
[make\\_test\\_response\(\)](#) (in module pyfarm.master.testutil), 81  
[mappings](#) (pyfarm.models.core.mixins.ModelTypes attribute), 86  
[MAX\\_CPUS](#) (pyfarm.models.agent.Agent attribute), 92  
[MAX\\_INT](#) (pyfarm.models.core.types.IPv4Address attribute), 88  
[MAX\\_INT](#) (pyfarm.models.core.types.MACAddress attribute), 89  
[MAX\\_PORT](#) (pyfarm.models.agent.Agent attribute), 92  
[MAX\\_PRIORITY](#) (pyfarm.models.core.mixins.ValidatePriorityMixin attribute), 87  
[MAX\\_RAM](#) (pyfarm.models.agent.Agent attribute), 92  
[maxDiff](#) (pyfarm.master.testutil.BaseTestCase attribute), 81  
[maximum\\_agents](#) (pyfarm.models.job.Job attribute), 97  
[maximum\\_agents](#) (pyfarm.models.jobqueue.JobQueue attribute), 100  
[methods](#) (pyfarm.master.api.agent\_updates.AgentUpdatesAPI attribute), 19  
[methods](#) (pyfarm.master.api.agents.AgentIndexAPI attribute), 21  
[methods](#) (pyfarm.master.api.agents.SingleAgentAPI attribute), 24  
[methods](#) (pyfarm.master.api.agents.SingleSoftwareInAgentAPI attribute), 25  
[methods](#) (pyfarm.master.api.agents.SoftwareInAgentIndexAPI attribute), 25  
[methods](#) (pyfarm.master.api.agents.TasksInAgentAPI attribute), 27  
[methods](#) (pyfarm.master.api.jobgroups.JobGroupIndexAPI attribute), 29  
[methods](#) (pyfarm.master.api.jobgroups.JobsInJobGroupIndexAPI attribute), 30  
[methods](#) (pyfarm.master.api.jobgroups.SingleJobGroupAPI attribute), 31  
[methods](#) (pyfarm.master.api.jobqueues.JobQueueIndexAPI attribute), 33  
[methods](#) (pyfarm.master.api.jobqueues.SingleJobQueueAPI attribute), 34  
[methods](#) (pyfarm.master.api.jobs.JobIndexAPI attribute), 36  
[methods](#) (pyfarm.master.api.jobs.JobNotifiedUsersIndexAPI attribute), 38  
[methods](#) (pyfarm.master.api.jobs.JobSingleNotifiedUserAPI attribute), 39  
[methods](#) (pyfarm.master.api.jobs.JobSingleTaskAPI attribute), 39  
[methods](#) (pyfarm.master.api.jobs.JobTasksIndexAPI attribute), 41  
[methods](#) (pyfarm.master.api.jobs.SingleJobAPI attribute), 43  
[methods](#) (pyfarm.master.api.jobs.SingleTaskOnAgentFailureAPI attribute), 45  
[methods](#) (pyfarm.master.api.jobs.TaskFailedOnAgentsIndexAPI attribute), 45  
[methods](#) (pyfarm.master.api.jobtypes.JobTypeCodeAPI attribute), 47  
[methods](#) (pyfarm.master.api.jobtypes.JobTypeIndexAPI attribute), 48  
[methods](#) (pyfarm.master.api.jobtypes.JobTypeSoftwareRequirementAPI attribute), 50  
[methods](#) (pyfarm.master.api.jobtypes.JobTypeSoftwareRequirementsIndexAPI attribute), 50  
[methods](#) (pyfarm.master.api.jobtypes.JobTypeVersionsIndexAPI attribute), 52  
[methods](#) (pyfarm.master.api.jobtypes.SingleJobTypeAPI attribute), 53  
[methods](#) (pyfarm.master.api.jobtypes.VersionedJobTypeAPI attribute), 55  
[methods](#) (pyfarm.master.api.pathmaps.PathMapIndexAPI attribute), 57  
[methods](#) (pyfarm.master.api.pathmaps.SinglePathMapAPI attribute), 58  
[methods](#) (pyfarm.master.api.software.SingleSoftwareAPI attribute), 60  
[methods](#) (pyfarm.master.api.software.SingleSoftwareVersionAPI attribute), 62  
[methods](#) (pyfarm.master.api.software.SoftwareIndexAPI attribute), 63  
[methods](#) (pyfarm.master.api.software.SoftwareVersionDiscoveryCodeAPI attribute), 64

- methods (pyfarm.master.api.software.SoftwareVersionsIndexAPI attribute), 64
  - methods (pyfarm.master.api.tags.AgentsInTagIndexAPI attribute), 66
  - methods (pyfarm.master.api.tags.SingleTagAPI attribute), 68
  - methods (pyfarm.master.api.tags.TagIndexAPI attribute), 70
  - methods (pyfarm.master.api.tasklogs.LogsInTaskAttemptsIndexAPI attribute), 72
  - methods (pyfarm.master.api.tasklogs.SingleLogInTaskAttemptsIndexAPI attribute), 73
  - methods (pyfarm.master.api.tasklogs.TaskLogfileAPI attribute), 74
  - MIN\_CPUS (pyfarm.models.agent.Agent attribute), 92
  - MIN\_PORT (pyfarm.models.agent.Agent attribute), 92
  - MIN\_PRIORITY (pyfarm.models.core.mixins.ValidatePriorityMixin attribute), 87
  - MIN\_RAM (pyfarm.models.agent.Agent attribute), 92
  - minimum\_agents (pyfarm.models.job.Job attribute), 97
  - minimum\_agents (pyfarm.models.jobqueue.JobQueue attribute), 100
  - modelfor() (in module pyfarm.models.core.functions), 85
  - ModelTypes (class in pyfarm.models.core.mixins), 86
  - mountpoint (pyfarm.models.disk.AgentDisk attribute), 95
  - move\_multiple\_jobs() (in module pyfarm.master.user\_interface.jobs), 76
- ## N
- name (pyfarm.models.jobqueue.JobQueue attribute), 100
  - name (pyfarm.models.jobtype.JobType attribute), 101
  - name (pyfarm.models.user.Role attribute), 107
  - NewUserForm (class in pyfarm.master.initial), 80
  - notes (pyfarm.models.agent.Agent attribute), 93
  - notes (pyfarm.models.job.Job attribute), 97
  - notified\_users (pyfarm.models.job.Job attribute), 97
  - num\_assigned\_agents() (pyfarm.models.job.Job method), 97
  - num\_assigned\_agents() (pyfarm.models.jobqueue.JobQueue method), 100
  - num\_deleted (pyfarm.models.statistics.task\_event\_count.TaskEventCount attribute), 91
  - num\_disabled (pyfarm.models.statistics.agent\_count.AgentCount attribute), 90
  - num\_done (pyfarm.models.statistics.task\_event\_count.TaskEventCount attribute), 91
  - num\_done\_tasks() (pyfarm.models.tasklog.TaskLog method), 105
  - num\_failed (pyfarm.models.statistics.task\_event\_count.TaskEventCount attribute), 91
  - num\_failed\_tasks() (pyfarm.models.tasklog.TaskLog method), 105
  - num\_new (pyfarm.models.statistics.task\_event\_count.TaskEventCount attribute), 91
  - num\_offline (pyfarm.models.statistics.agent\_count.AgentCount attribute), 90
  - num\_online (pyfarm.models.statistics.agent\_count.AgentCount attribute), 91
  - num\_queued\_tasks() (pyfarm.models.tasklog.TaskLog method), 105
  - num\_started (pyfarm.models.statistics.task\_event\_count.TaskEventCount attribute), 92
  - num\_running (pyfarm.models.statistics.agent\_count.AgentCount attribute), 91
  - num\_running\_tasks() (pyfarm.models.tasklog.TaskLog method), 105
  - num\_started (pyfarm.models.statistics.task\_event\_count.TaskEventCount attribute), 92
  - num\_tasks (pyfarm.models.job.Job attribute), 97
- ## O
- ObjectNotFound, 41, 52
  - onetime\_code (pyfarm.models.user.User attribute), 106
  - OperatingSystemEnum (class in pyfarm.models.core.types), 89
  - ORIGINAL\_ENVIRONMENT (pyfarm.master.testutil.BaseTestCase attribute), 80
  - os\_class (pyfarm.models.agent.Agent attribute), 93
  - os\_fullname (pyfarm.models.agent.Agent attribute), 93
  - output\_link (pyfarm.models.job.Job attribute), 97
- ## P
- parent (pyfarm.models.jobqueue.JobQueue attribute), 100
  - parent\_jobqueue\_id (pyfarm.models.jobqueue.JobQueue attribute), 100
  - parents (pyfarm.models.job.Job attribute), 97
  - parse\_requirements() (in module pyfarm.master.api.jobs), 46
  - parse\_requirements() (in module pyfarm.master.api.jobtypes), 55
  - password (pyfarm.master.initial.NewUserForm attribute), 80
  - password (pyfarm.models.user.User attribute), 106
  - path() (pyfarm.models.jobqueue.JobQueue method), 100
  - path\_linux (pyfarm.models.pathmap.PathMap attribute), 101
  - path\_linux (pyfarm.models.pathmap.PathMap attribute), 101
  - path\_windows (pyfarm.models.pathmap.PathMap attribute), 102
  - PathMap (class in pyfarm.models.pathmap), 101
  - PathMapIndexAPI (class in pyfarm.master.api.pathmaps), 56
  - pause\_multiple\_jobs() (in module pyfarm.master.user\_interface.jobs), 76

[pause\\_single\\_job\(\)](#) (in module `pyfarm.master.user_interface.jobs`), 76  
[paused\(\)](#) (`pyfarm.models.job.Job` method), 97  
[port](#) (`pyfarm.models.agent.Agent` attribute), 94  
[post\(\)](#) (`pyfarm.master.api.agents.AgentIndexAPI` method), 21  
[post\(\)](#) (`pyfarm.master.api.agents.SingleAgentAPI` method), 24  
[post\(\)](#) (`pyfarm.master.api.agents.SoftwareInAgentIndexAPI` method), 25  
[post\(\)](#) (`pyfarm.master.api.agents.TasksInAgentAPI` method), 27  
[post\(\)](#) (`pyfarm.master.api.jobgroups.JobGroupIndexAPI` method), 29  
[post\(\)](#) (`pyfarm.master.api.jobgroups.SingleJobGroupAPI` method), 31  
[post\(\)](#) (`pyfarm.master.api.jobqueues.JobQueueIndexAPI` method), 33  
[post\(\)](#) (`pyfarm.master.api.jobqueues.SingleJobQueueAPI` method), 34  
[post\(\)](#) (`pyfarm.master.api.jobs.JobIndexAPI` method), 36  
[post\(\)](#) (`pyfarm.master.api.jobs.JobNotifiedUsersIndexAPI` method), 38  
[post\(\)](#) (`pyfarm.master.api.jobs.JobSingleTaskAPI` method), 40  
[post\(\)](#) (`pyfarm.master.api.jobs.SingleJobAPI` method), 43  
[post\(\)](#) (`pyfarm.master.api.jobs.TaskFailedOnAgentsIndexAPI` method), 45  
[post\(\)](#) (`pyfarm.master.api.jobtypes.JobTypeIndexAPI` method), 48  
[post\(\)](#) (`pyfarm.master.api.jobtypes.JobTypeSoftwareRequirementsIndexAPI` method), 50  
[post\(\)](#) (`pyfarm.master.api.pathmaps.PathMapIndexAPI` method), 57  
[post\(\)](#) (`pyfarm.master.api.pathmaps.SinglePathMapAPI` method), 58  
[post\(\)](#) (`pyfarm.master.api.software.SoftwareIndexAPI` method), 63  
[post\(\)](#) (`pyfarm.master.api.software.SoftwareVersionsIndexAPI` method), 64  
[post\(\)](#) (`pyfarm.master.api.tags.AgentsInTagIndexAPI` method), 66  
[post\(\)](#) (`pyfarm.master.api.tags.TagIndexAPI` method), 70  
[post\(\)](#) (`pyfarm.master.api.tasklogs.LogsInTaskAttemptsIndexAPI` method), 72  
[post\(\)](#) (`pyfarm.master.api.tasklogs.SingleLogInTaskAttemptIndexAPI` method), 73  
[primary\\_keys](#) (`pyfarm.models.core.mixins.ModelTypes` attribute), 86  
[priority](#) (`pyfarm.models.job.Job` attribute), 97  
[priority](#) (`pyfarm.models.jobqueue.JobQueue` attribute), 100  
[priority](#) (`pyfarm.models.task.Task` attribute), 104  
[process\\_bind\\_param\(\)](#) (`pyfarm.models.core.types.EnumType` method), 88  
[process\\_bind\\_param\(\)](#) (`pyfarm.models.core.types.IPv4Address` method), 88  
[process\\_bind\\_param\(\)](#) (`pyfarm.models.core.types.JSONSerializable` method), 89  
[process\\_bind\\_param\(\)](#) (`pyfarm.models.core.types.MACAddress` method), 89  
[process\\_bind\\_param\(\)](#) (`pyfarm.models.core.types.UUIDType` method), 90  
[process\\_result\\_value\(\)](#) (`pyfarm.models.core.types.EnumType` method), 88  
[process\\_result\\_value\(\)](#) (`pyfarm.models.core.types.IPv4Address` method), 88  
[process\\_result\\_value\(\)](#) (`pyfarm.models.core.types.JSONSerializable` method), 89  
[process\\_result\\_value\(\)](#) (`pyfarm.models.core.types.MACAddress` method), 89  
[process\\_result\\_value\(\)](#) (`pyfarm.models.core.types.UUIDType` method), 90  
[progress](#) (`pyfarm.models.task.Task` attribute), 104  
[put\(\)](#) (`pyfarm.master.api.agent_updates.AgentUpdatesAPI` method), 19  
[put\(\)](#) (`pyfarm.master.api.jobtypes.SingleJobTypeAPI` method), 53  
[put\(\)](#) (`pyfarm.master.api.software.SingleSoftwareAPI` method), 60  
[put\(\)](#) (`pyfarm.master.api.tags.SingleTagAPI` method), 68  
[put\(\)](#) (`pyfarm.master.api.tasklogs.TaskLogfileAPI` method), 74  
[pyfarm.master](#) (module), 84  
[pyfarm.master.api](#) (module), 74  
[pyfarm.master.api.agent\\_updates](#) (module), 19  
[pyfarm.master.api.agents](#) (module), 20  
[pyfarm.master.api.jobgroups](#) (module), 28  
[pyfarm.master.api.jobqueues](#) (module), 32  
[pyfarm.master.api.jobs](#) (module), 35  
[pyfarm.master.api.jobtypes](#) (module), 47  
[pyfarm.master.api.pathmaps](#) (module), 56  
[pyfarm.master.api.software](#) (module), 59  
[pyfarm.master.api.tags](#) (module), 66  
[pyfarm.master.api.tasklogs](#) (module), 71  
[pyfarm.master.application](#) (module), 78  
[pyfarm.master.config](#) (module), 79



pyfarm.master.entrypoints (module), 79  
 pyfarm.master.index (module), 80  
 pyfarm.master.initial (module), 80  
 pyfarm.master.testutil (module), 80  
 pyfarm.master.user\_interface (module), 78  
 pyfarm.master.user\_interface.agents (module), 75  
 pyfarm.master.user\_interface.jobgroups (module), 76  
 pyfarm.master.user\_interface.jobqueues (module), 76  
 pyfarm.master.user\_interface.jobs (module), 76  
 pyfarm.master.user\_interface.jobtypes (module), 77  
 pyfarm.master.user\_interface.software (module), 77  
 pyfarm.master.user\_interface.software\_version (module), 77  
 pyfarm.master.user\_interface.statistics (module), 75  
 pyfarm.master.user\_interface.statistics.agent\_counts (module), 75  
 pyfarm.master.user\_interface.statistics.index (module), 75  
 pyfarm.master.user\_interface.statistics.task\_events (module), 75  
 pyfarm.master.utility (module), 82  
 pyfarm.models (module), 107  
 pyfarm.models.agent (module), 92  
 pyfarm.models.core (module), 90  
 pyfarm.models.core.functions (module), 85  
 pyfarm.models.core.mixins (module), 86  
 pyfarm.models.core.types (module), 88  
 pyfarm.models.disk (module), 95  
 pyfarm.models.gpu (module), 95  
 pyfarm.models.job (module), 96  
 pyfarm.models.jobgroup (module), 99  
 pyfarm.models.jobqueue (module), 99  
 pyfarm.models.jobtype (module), 100  
 pyfarm.models.pathmap (module), 101  
 pyfarm.models.software (module), 102  
 pyfarm.models.statistics (module), 92  
 pyfarm.models.statistics.agent\_count (module), 90  
 pyfarm.models.statistics.task\_count (module), 91  
 pyfarm.models.statistics.task\_event\_count (module), 91  
 pyfarm.models.tag (module), 102  
 pyfarm.models.task (module), 103  
 pyfarm.models.tasklog (module), 104  
 pyfarm.models.user (module), 105  
 pyfarm.scheduler (module), 109  
 pyfarm.scheduler.celery\_app (module), 109  
 pyfarm.scheduler.statistics\_tasks (module), 109  
 pyfarm.scheduler.tasks (module), 109

## Q

queue (pyfarm.models.job.Job attribute), 97

## R

ram (pyfarm.models.agent.Agent attribute), 94  
 ram (pyfarm.models.job.Job attribute), 97

ram\_allocation (pyfarm.models.agent.Agent attribute), 94  
 ram\_max (pyfarm.models.job.Job attribute), 98  
 ram\_warning (pyfarm.models.job.Job attribute), 98  
 relationships (pyfarm.models.core.mixins.ModelTypes attribute), 86  
 remote\_ip (pyfarm.models.agent.Agent attribute), 94  
 remove\_jobtype() (in module pyfarm.master.user\_interface.jobtypes), 77  
 remove\_jobtype\_software\_requirement() (in module pyfarm.master.user\_interface.jobtypes), 77  
 remove\_notified\_user\_from\_job() (in module pyfarm.master.user\_interface.jobs), 76  
 remove\_software() (in module pyfarm.master.user\_interface.software), 77  
 remove\_software\_version() (in module pyfarm.master.user\_interface.software), 77  
 remove\_tag\_from\_jobs() (in module pyfarm.master.user\_interface.jobs), 76  
 remove\_tag\_requirement\_from\_jobs() (in module pyfarm.master.user\_interface.jobs), 76  
 REPR\_COLUMNS (pyfarm.models.agent.Agent attribute), 92  
 REPR\_COLUMNS (pyfarm.models.core.mixins.ReprMixin attribute), 86  
 REPR\_COLUMNS (pyfarm.models.job.Job attribute), 96  
 REPR\_COLUMNS (pyfarm.models.jobqueue.JobQueue attribute), 99  
 REPR\_COLUMNS (pyfarm.models.jobtype.JobType attribute), 101  
 REPR\_COLUMNS (pyfarm.models.task.Task attribute), 103  
 REPR\_COLUMNS (pyfarm.models.user.User attribute), 105  
 REPR\_CONVERT\_COLUMN (pyfarm.models.agent.Agent attribute), 92  
 REPR\_CONVERT\_COLUMN (pyfarm.models.core.mixins.ReprMixin attribute), 86  
 REPR\_CONVERT\_COLUMN (pyfarm.models.job.Job attribute), 96  
 REPR\_CONVERT\_COLUMN (pyfarm.models.task.Task attribute), 103  
 repr\_enum() (in module pyfarm.models.core.functions), 85  
 repr\_ip() (in module pyfarm.models.core.functions), 86  
 ReprMixin (class in pyfarm.models.core.mixins), 86  
 requeue (pyfarm.models.job.Job attribute), 98  
 required (pyfarm.models.core.mixins.ModelTypes attribute), 86  
 rerun() (pyfarm.models.job.Job method), 98  
 rerun\_failed() (pyfarm.models.job.Job method), 98  
 rerun\_failed\_in\_job() (in module pyfarm.master.user\_interface.jobs), 76

rerun\_failed\_in\_multiple\_jobs() (in module pyfarm.master.user\_interface.jobs), 76  
 rerun\_multiple\_jobs() (in module pyfarm.master.user\_interface.jobs), 76  
 rerun\_single\_job() (in module pyfarm.master.user\_interface.jobs), 76  
 rerun\_single\_task() (in module pyfarm.master.user\_interface.jobs), 76  
 reset\_agent\_if\_failed\_and\_retry() (pyfarm.models.task.Task static method), 104  
 reset\_finished\_time() (pyfarm.models.task.Task static method), 104  
 restart\_multiple\_agents() (in module pyfarm.master.user\_interface.agents), 75  
 restart\_requested (pyfarm.models.agent.Agent attribute), 94  
 restart\_single\_agent() (in module pyfarm.master.user\_interface.agents), 75  
 RFC  
     RFC 1112, 94  
     RFC 1918, 95  
     RFC 3927, 94  
     RFC 4632, 94  
     RFC 6052, 95  
 Role (class in pyfarm.models.user), 106  
 roles (pyfarm.models.user.User attribute), 106  
 run\_master() (in module pyfarm.master.entrypoints), 79  
 running() (pyfarm.models.task.Task method), 104  
**S**  
 satisfies\_job\_requirements() (pyfarm.models.agent.Agent method), 94  
 satisfies\_jobtype\_requirements() (pyfarm.models.agent.Agent method), 94  
 schema() (in module pyfarm.master.api.agents), 28  
 schema() (in module pyfarm.master.api.jobgroups), 31  
 schema() (in module pyfarm.master.api.jobqueues), 35  
 schema() (in module pyfarm.master.api.jobs), 46  
 schema() (in module pyfarm.master.api.jobtypes), 55  
 schema() (in module pyfarm.master.api.pathmaps), 59  
 schema() (in module pyfarm.master.api.software), 65  
 schema() (in module pyfarm.master.api.tags), 71  
 send\_email() (in module pyfarm.scheduler.tasks), 109  
 sent\_to\_agent (pyfarm.models.task.Task attribute), 104  
 serialize\_none (pyfarm.models.core.types.JSONSerializable attribute), 89  
 serialize\_types (pyfarm.models.core.types.JSONDict attribute), 89  
 serialize\_types (pyfarm.models.core.types.JSONList attribute), 89  
 serialize\_types (pyfarm.models.core.types.JSONSerializable attribute), 89  
 SessionMixin (class in pyfarm.master.application), 78  
 set\_prio\_weight\_on\_jobs() (in module pyfarm.master.user\_interface.jobs), 76  
 set\_progress\_on\_success() (pyfarm.models.task.Task static method), 104  
 set\_times() (pyfarm.models.task.Task static method), 104  
 setUp() (pyfarm.master.testutil.BaseTestCase method), 81  
 setup\_app() (pyfarm.master.testutil.BaseTestCase method), 81  
 setup\_client() (pyfarm.master.testutil.BaseTestCase method), 81  
 setup\_database() (pyfarm.master.testutil.BaseTestCase method), 81  
 setup\_page() (in module pyfarm.master.initial), 80  
 setup\_warning\_filter() (pyfarm.master.testutil.BaseTestCase method), 81  
 single\_agent() (in module pyfarm.master.user\_interface.agents), 75  
 single\_job() (in module pyfarm.master.user\_interface.jobs), 76  
 SingleAgentAPI (class in pyfarm.master.api.agents), 22  
 SingleJobAPI (class in pyfarm.master.api.jobs), 41  
 SingleJobGroupAPI (class in pyfarm.master.api.jobgroups), 30  
 SingleJobQueueAPI (class in pyfarm.master.api.jobqueues), 33  
 SingleJobTypeAPI (class in pyfarm.master.api.jobtypes), 52  
 SingleLogInTaskAttempt (class in pyfarm.master.api.tasklogs), 72  
 SinglePathMapAPI (class in pyfarm.master.api.pathmaps), 57  
 SingleSoftwareAPI (class in pyfarm.master.api.software), 59  
 SingleSoftwareInAgentAPI (class in pyfarm.master.api.agents), 24  
 SingleSoftwareVersionAPI (class in pyfarm.master.api.software), 61  
 SingleTagAPI (class in pyfarm.master.api.tags), 67  
 SingleTaskOnAgentFailureAPI (class in pyfarm.master.api.jobs), 44  
 size (pyfarm.models.disk.AgentDisk attribute), 95  
 Software (class in pyfarm.models.software), 102  
 software (pyfarm.models.software.Software attribute), 102  
 software() (in module pyfarm.master.user\_interface.software), 77  
 software\_item() (in module pyfarm.master.user\_interface.software), 77  
 software\_requirements (pyfarm.models.job.Job attribute), 98  
 software\_version() (in module pyfarm.master.user\_interface.software\_version),

- 77
- software\_versions (pyfarm.models.agent.Agent attribute), 94
- SoftwareInAgentIndexAPI (class in pyfarm.master.api.agents), 25
- SoftwareIndexAPI (class in pyfarm.master.api.software), 62
- SoftwareVersionDiscoveryCodeAPI (class in pyfarm.master.api.software), 63
- SoftwareVersionsIndexAPI (class in pyfarm.master.api.software), 64
- split\_and\_extend() (in module pyfarm.models.core.functions), 86
- state (pyfarm.models.agent.Agent attribute), 94
- state (pyfarm.models.job.Job attribute), 98
- state (pyfarm.models.task.Task attribute), 104
- state (pyfarm.models.tasklog.TaskTaskLogAssociation attribute), 105
- state\_changed() (pyfarm.models.core.mixins.WorkStateChangeMixin static method), 87
- STATE\_DEFAULT (pyfarm.models.agent.Agent attribute), 92
- STATE\_DEFAULT (pyfarm.models.task.Task attribute), 103
- STATE\_ENUM (pyfarm.models.agent.Agent attribute), 92
- STATE\_ENUM (pyfarm.models.core.mixins.ValidateWorkStateMixin attribute), 87
- STATE\_ENUM (pyfarm.models.job.Job attribute), 96
- STATE\_ENUM (pyfarm.models.task.Task attribute), 103
- statistics\_index() (in module pyfarm.master.user\_interface.statistics.index), 75
- subscribed\_jobs (pyfarm.models.user.User attribute), 106
- success\_body (pyfarm.models.jobtype.JobType attribute), 101
- success\_subject (pyfarm.models.jobtype.JobType attribute), 101
- T**
- tables() (in module pyfarm.master.entrypoints), 79
- Tag (class in pyfarm.models.tag), 102
- tag (pyfarm.models.pathmap.PathMap attribute), 102
- tag (pyfarm.models.tag.Tag attribute), 102
- tag\_id (pyfarm.models.pathmap.PathMap attribute), 102
- tag\_requirements (pyfarm.models.job.Job attribute), 98
- TagIndexAPI (class in pyfarm.master.api.tags), 69
- tags (pyfarm.models.agent.Agent attribute), 94
- tags (pyfarm.models.job.Job attribute), 98
- Task (class in pyfarm.models.task), 103
- task (pyfarm.models.tasklog.TaskTaskLogAssociation attribute), 105
- task\_associations (pyfarm.models.tasklog.TaskLog attribute), 105
- task\_events() (in module pyfarm.master.user\_interface.statistics.task\_events), 75
- task\_id (pyfarm.models.tasklog.TaskTaskLogAssociation attribute), 105
- task\_log\_id (pyfarm.models.tasklog.TaskTaskLogAssociation attribute), 105
- task\_logs (pyfarm.models.agent.Agent attribute), 94
- TaskCount (class in pyfarm.models.statistics.task\_count), 91
- TaskEventCount (class in pyfarm.models.statistics.task\_event\_count), 91
- TaskFailedOnAgentsIndexAPI (class in pyfarm.master.api.jobs), 45
- TaskLog (class in pyfarm.models.tasklog), 104
- TaskLogfileAPI (class in pyfarm.master.api.tasklogs), 73
- tasks (pyfarm.models.agent.Agent attribute), 94
- tasks (pyfarm.models.job.Job attribute), 98
- tasks\_done (pyfarm.models.job.Job attribute), 98
- tasks\_failed (pyfarm.models.job.Job attribute), 98
- tasks\_queued (pyfarm.models.job.Job attribute), 98
- tasks\_running (pyfarm.models.job.Job attribute), 98
- TasksInAgentAPI (class in pyfarm.master.api.agents), 26
- TaskTaskLogAssociation (class in pyfarm.models.tasklog), 105
- tearDown() (pyfarm.master.testutil.BaseTestCase method), 81
- tearDown\_app() (pyfarm.master.testutil.BaseTestCase method), 81
- tearDown\_database() (pyfarm.master.testutil.BaseTestCase method), 81
- tearDown\_warning\_filter() (pyfarm.master.testutil.BaseTestCase method), 81
- tile (pyfarm.models.task.Task attribute), 104
- time\_end (pyfarm.models.statistics.task\_event\_count.TaskEventCount attribute), 92
- time\_finished (pyfarm.models.job.Job attribute), 98
- time\_finished (pyfarm.models.task.Task attribute), 104
- time\_offset (pyfarm.models.agent.Agent attribute), 94
- time\_start (pyfarm.models.statistics.task\_event\_count.TaskEventCount attribute), 92
- time\_started (pyfarm.models.job.Job attribute), 98
- time\_started (pyfarm.models.task.Task attribute), 104
- time\_submitted (pyfarm.models.job.Job attribute), 98
- time\_submitted (pyfarm.models.task.Task attribute), 104
- timedelta\_format() (in module pyfarm.master.utility), 83
- title (pyfarm.models.job.Job attribute), 98
- title (pyfarm.models.jobgroup.JobGroup attribute), 99
- to\_be\_deleted (pyfarm.models.job.Job attribute), 98
- to\_dict() (pyfarm.models.core.mixins.UtilityMixins method), 87

- to\_python() (pyfarm.master.application.UUIDConverter method), 78
  - to\_schema() (pyfarm.models.core.mixins.UtilityMixins class method), 87
  - to\_url() (pyfarm.master.application.UUIDConverter method), 78
  - top\_level\_unique\_check() (pyfarm.models.jobqueue.JobQueue static method), 100
  - total\_done (pyfarm.models.statistics.task\_count.TaskCount attribute), 91
  - total\_failed (pyfarm.models.statistics.task\_count.TaskCount attribute), 91
  - total\_queued (pyfarm.models.statistics.task\_count.TaskCount attribute), 91
  - total\_running (pyfarm.models.statistics.task\_count.TaskCount attribute), 91
  - TotalsAverage (class in pyfarm.master.user\_interface.statistics.task\_events), 75
  - types() (pyfarm.models.core.mixins.UtilityMixins class method), 87
- ## U
- unpause\_multiple\_jobs() (in module pyfarm.master.user\_interface.jobs), 76
  - unpause\_single\_job() (in module pyfarm.master.user\_interface.jobs), 76
  - update\_agent\_on\_success() (pyfarm.models.task.Task static method), 104
  - update\_failures() (pyfarm.models.task.Task static method), 104
  - update\_jobtype\_notification\_templates() (in module pyfarm.master.user\_interface.jobtypes), 77
  - update\_notes\_for\_agent() (in module pyfarm.master.user\_interface.agents), 75
  - update\_notes\_for\_job() (in module pyfarm.master.user\_interface.jobs), 77
  - update\_state() (pyfarm.models.job.Job method), 98
  - update\_tag\_requirements\_in\_job() (in module pyfarm.master.user\_interface.jobs), 77
  - update\_tags\_in\_agent() (in module pyfarm.master.user\_interface.agents), 75
  - update\_tags\_in\_job() (in module pyfarm.master.user\_interface.jobs), 77
  - update\_version\_default\_status() (in module pyfarm.master.user\_interface.software), 77
  - update\_version\_rank() (in module pyfarm.master.user\_interface.software), 77
  - upgrade\_job\_to\_latest\_jobtype\_version() (in module pyfarm.master.user\_interface.jobs), 77
  - upgrade\_to (pyfarm.models.agent.Agent attribute), 94
  - URL\_TEMPLATE (pyfarm.models.agent.Agent attribute), 92
  - use\_address (pyfarm.models.agent.Agent attribute), 94
  - UseAgentAddressEnum (class in pyfarm.models.core.types), 90
  - User (class in pyfarm.models.user), 105
  - user (pyfarm.models.job.Job attribute), 98
  - user (pyfarm.models.jobgroup.JobGroup attribute), 99
  - user\_id (pyfarm.models.job.Job attribute), 99
  - user\_id (pyfarm.models.jobgroup.JobGroup attribute), 99
  - username (pyfarm.master.initial.NewUserForm attribute), 80
  - username (pyfarm.models.user.User attribute), 106
  - users (pyfarm.models.user.Role attribute), 107
  - UtilityMixins (class in pyfarm.models.core.mixins), 86
  - UUIDConverter (class in pyfarm.master.application), 78
  - UUIDType (class in pyfarm.models.core.types), 90
- ## V
- validate\_attempts() (pyfarm.models.core.mixins.ValidatePriorityMixin method), 87
  - validate\_hostname() (pyfarm.models.agent.Agent class method), 94
  - validate\_hostname\_column() (pyfarm.models.agent.Agent method), 94
  - validate\_ipv4\_address() (pyfarm.models.agent.Agent class method), 94
  - validate\_json() (in module pyfarm.master.utility), 83
  - validate\_name() (pyfarm.models.jobtype.JobType method), 101
  - validate\_numeric\_column() (pyfarm.models.agent.Agent method), 95
  - validate\_priority() (pyfarm.models.core.mixins.ValidatePriorityMixin method), 87
  - validate\_progress() (pyfarm.models.job.Job method), 99
  - validate\_remote\_ip() (pyfarm.models.agent.Agent method), 95
  - validate\_resource() (pyfarm.models.agent.Agent class method), 95
  - validate\_resource() (pyfarm.models.job.Job method), 99
  - validate\_state() (pyfarm.models.core.mixins.ValidateWorkStateMixin method), 87
  - validate\_state\_column() (pyfarm.models.core.mixins.ValidateWorkStateMixin method), 87
  - validate\_username() (pyfarm.master.initial.NewUserForm method), 80
  - validate\_with\_model() (in module pyfarm.master.utility), 84
  - ValidatePriorityMixin (class in pyfarm.models.core.mixins), 87
  - ValidateWorkStateMixin (class in pyfarm.models.core.mixins), 87
  - version (pyfarm.models.agent.Agent attribute), 95

VersionedJobTypeAPI (class in pyfarm.master.api.jobtypes), 54

VersionParseError, 65

versions (pyfarm.models.jobtype.JobType attribute), 101

versions (pyfarm.models.software.Software attribute), 102

## W

weight (pyfarm.models.job.Job attribute), 99

weight (pyfarm.models.jobqueue.JobQueue attribute), 100

work\_columns() (in module pyfarm.models.core.functions), 86

WorkStateChangedMixin (class in pyfarm.models.core.mixins), 87

WorkStateEnum (class in pyfarm.models.core.types), 90