
pyfarm.agent Documentation

Release 0.8.7

Oliver Palmer, Guido Winkelmann

Dec 22, 2017

Contents

1	Commands	3
1.1	Standard Commands	3
1.2	Development Commands	8
2	Environment Variables	11
3	Configuration Files	13
3.1	Agent	13
3.2	Job Types	17
4	pyfarm.agent package	19
4.1	Subpackages	19
4.2	Submodules	36
4.3	Module contents	44
5	pyfarm.jobtypes package	45
5.1	Subpackages	45
5.2	Module contents	57
6	Indices and tables	59
	HTTP Routing Table	61
	Python Module Index	63

This package contains PyFarm's agent and job types which are responsible for the execution of tasks allocated to a host by the master.

Contents

Note: The default values provided are based on the configuration at the time this page was generated. They may not be the same defaults you will see.

1.1 Standard Commands

1.1.1 pyfarm-agent

```
usage: pyfarm-agent [status|start|stop]

positional arguments:
  {start,stop,status}  individual operations pyfarm-agent can run
    start              starts the agent
    stop               stops the agent
    status              query the 'running' state of the agent

optional arguments:
  -h, --help            show this help message and exit

Agent Network Service:
  Main flags which control the network services running on the agent.

  --port PORT           The port number which the agent is either running on
                        or will run on when started. This port is also
                        reported the master when an agent starts. [default:
                        None]
  --host HOST           The host to communicate with or hostname to present to
                        the master when starting. Defaults to the fully
                        qualified hostname.
  --agent-api-username AGENT_API_USERNAME
                        The username required to access or manipulate the
```

```
agent using REST. [default: agent]
--agent-api-password AGENT_API_PASSWORD
    The password required to access manipulate the agent
    using REST. [default: agent]
--agent-id AGENT_ID
    The UUID used to identify this agent to the master. By
    default the agent will attempt to load a cached value
    however a specific UUID could be provided with this
    flag.
--agent-id-file AGENT_ID_FILE
    The location to store the agent's id. By default the
    path is platform specific and defined by the
    `agent_id_file_platform_defaults` key in the
    configuration. [default: /etc/pyfarm/agent/uuid.dat]
```

Network Resources:

Resources which the agent will be communicating with.

```
--master MASTER
    This is a convenience flag which will allow you to set
    the hostname for the master. By default this value
    will be substituted in --master-api
--master-api MASTER_API
    The location where the master's REST api is located.
    [default: None]
--master-api-version MASTER_API_VERSION
    Sets the version of the master's REST api the agent
    should use [default: None]
```

Process Control:

These settings apply to the parent process of the agent and contribute to allowing the process to run as other users or remain isolated in an environment. They also assist in maintaining the 'running state' via a process id file.

```
--pidfile PIDFILE
    The file to store the process id in. [default: None]
-n, --no-daemon
    If provided then do not run the process in the
    background.
--chdir CHDIR
    The working directory to change the agent into upon
    launch
--uid UID
    The user id to run the agent as. *This setting is
    ignored on Windows.*
--gid GID
    The group id to run the agent as. *This setting is
    ignored on Windows.*
--pdb-on-unhandled
    When set pdb.set_trace() will be called if an
    unhandled error is caught in the logger
```

pyfarm-agent is a command line client for working with a local agent. You can use it to stop, start, and report the general status of a running agent process.

```
usage: pyfarm-agent [status|start|stop] status [-h]
```

optional arguments:

```
-h, --help show this help message and exit
```

```
usage: pyfarm-agent [status|start|stop] start [-h] [--state STATE]
                                         [--time-offset TIME_OFFSET]
                                         [--ntp-server NTP_SERVER]
```

```

↪VERSION]
                                [--ntp-server-version NTP_SERVER_
                                [--no-pretty-json]
                                [--shutdown-timeout SHUTDOWN_TIMEOUT]
                                [--updates-drop-dir UPDATES_DROP_DIR]
                                [--run-control-file RUN_CONTROL_FILE]
                                [--farm-name FARM_NAME]
                                [--cpus CPUS] [--ram RAM]
                                [--ram-check-interval RAM_CHECK_
↪INTERVAL]
                                [--ram-max-report-frequency RAM_MAX_
↪REPORT_FREQUENCY]
                                [--ram-report-delta RAM_REPORT_DELTA]
                                [--master-reannounce MASTER_REANNOUNCE]
                                [--log LOG]
                                [--capture-process-output]
                                [--task-log-dir TASK_LOG_DIR]
                                [--ip-remote IP_REMOTE]
                                [--enable-manhole]
                                [--manhole-port MANHOLE_PORT]
                                [--manhole-username MANHOLE_USERNAME]
                                [--manhole-password MANHOLE_PASSWORD]
                                [--html-templates-reload]
                                [--static-files STATIC_FILES]
                                [--http-retry-delay-offset HTTP_RETRY_
↪DELAY_OFFSET]
                                [--http-retry-delay-factor HTTP_RETRY_
↪DELAY_FACTOR]
                                [--jobtype-no-cache]

```

optional arguments:

```
-h, --help          show this help message and exit
```

General Configuration:

These flags configure parts of the agent related to hardware, state, and certain timing and scheduling attributes.

```

--state STATE          The current agent state, valid values are ['disabled',
                        'offline', 'running', 'online']. [default: online]
--time-offset TIME_OFFSET
                        If provided then don't talk to the NTP server at all
                        to calculate the time offset. If you know for a fact
                        that this host's time is always up to date then
                        setting this to 0 is probably a safe bet.
--ntp-server NTP_SERVER
                        The default network time server this agent should
                        query to retrieve the real time. This will be used to
                        help determine the agent's clock skew if any. Setting
                        this value to '' will effectively disable this query.
                        [default: None]
--ntp-server-version NTP_SERVER_VERSION
                        The version of the NTP server in case it's running an
                        older or newer version. [default: None]
--no-pretty-json       If provided do not dump human readable json via the
                        agent's REST api
--shutdown-timeout SHUTDOWN_TIMEOUT
                        How many seconds the agent should spend attempting to
                        inform the master that it's shutting down.

```

```
--updates-drop-dir UPDATES_DROP_DIR
    The directory to drop downloaded updates in. This
    should be the same directory pyfarm-supervisor will
    look for updates in. [default: None]
--run-control-file RUN_CONTROL_FILE
    The path to a file that will signal to the supervisor
    that agent is supposed to be restarted if it stops for
    whatever reason. [default:
    /tmp/pyfarm/agent/should_be_running]
--farm-name FARM_NAME
    The name of the farm the agent should join. If unset,
    the agent will join any farm.
```

Physical Hardware:

Command line flags which describe the hardware of the agent.

```
--cpus CPUS
    The total amount of cpus installed on the system.
    Defaults to the number of cpus installed on the
    system.
--ram RAM
    The total amount of ram installed on the system in
    megabytes. Defaults to the amount of ram the system
    has installed.
```

Interval Controls:

Controls which dictate when certain internal intervals should occur.

```
--ram-check-interval RAM_CHECK_INTERVAL
    How often ram resources should be checked for changes.
    The amount of memory currently being consumed on the
    system is checked after certain events occur such as a
    process but this flag specifically controls how often
    we should check when no such events are occurring.
    [default: None]
--ram-max-report-frequency RAM_MAX_REPORT_FREQUENCY
    This is a limiter that prevents the agent from
    reporting memory changes to the master more often than
    a specific time interval. This is done in order to
    ensure that when 100s of events fire in a short period
    of time cause changes in ram usage only one or two
    will be reported to the master. [default: None]
--ram-report-delta RAM_REPORT_DELTA
    Only report a change in ram if the value has changed
    at least this many megabytes. [default: None]
--master-reannounce MASTER_REANNOUNCE
    Controls how often the agent should reannounce itself
    to the master. The agent may be in contact with the
    master more often than this however during long period
    of inactivity this is how often the agent will
    'inform' the master the agent is still online.
```

Logging Options:

Settings which control logging of the agent's **parent process and/or any** subprocess it runs.

```
--log LOG
    If provided log all output from the agent to this
    path. This will append to any existing log data.
    [default: None]
--capture-process-output
```

If provided then all log output from each process launched by the agent will be sent through agent's loggers.

--task-log-dir TASK_LOG_DIR

The directory tasks should log to.

Network Service:

Controls how the agent is seen or interacted with by external services such as the master.

--ip-remote IP_REMOTE

The remote IPv4 address to report. In situation where the agent is behind a firewall this value will typically be different.

Manhole Service:

Controls the manhole service which allows a telnet connection to be made directly into the agent as it's running.

--enable-manhole When provided the manhole service will be started once the reactor is running.

--manhole-port MANHOLE_PORT

The port the manhole service should run on if enabled.

--manhole-username MANHOLE_USERNAME

The telnet username that's allowed to connect to the manhole service running on the agent.

--manhole-password MANHOLE_PASSWORD

The telnet password to use when connecting to the manhole service running on the agent.

HTTP Configuration:

Options for how the agent will interact with the master's REST api and how it should run it's own REST api.

--html-templates-reload

If provided then force Jinja2, the html template system, to check the file system for changes with every request. This flag should not be used in production but is useful for development and debugging purposes.

--static-files STATIC_FILES

The default location where the agent's http server should find static files to serve.

--http-retry-delay-offset HTTP_RETRY_DELAY_OFFSET

If a http request to the master has failed, wait at least this amount of time before resending the request.

--http-retry-delay-factor HTTP_RETRY_DELAY_FACTOR

The value provided here is used in combination with --http-retry-delay-offset to calculate the retry delay. This is used as a multiplier against random() before being added to the offset.

Job Types:

--jobtype-no-cache

If provided then do not cache job types, always directly retrieve them. This is beneficial if you're testing the agent or a new job type class.

```
usage: pyfarm-agent [status|start|stop] stop [-h] [--no-wait]
```

optional arguments:

-h, --help show this help message **and** exit

optional flags:

Flags that control how the agent **is** stopped

--no-wait If provided then don't wait on the agent to shut itself down. By default we would want to wait on each task to stop so we can catch **any** errors **and** then **finally** wait on the agent to shutdown too. If you're in a hurry or stopping a bunch of agents at once then setting this flag will let the agent **continue** to stop itself without waiting **for** each agent

```
usage: pyfarm-supervisor [-h] [--updates-drop-dir UPDATES_DROP_DIR]
                        [--agent-package-dir AGENT_PACKAGE_DIR]
                        [--pidfile PIDFILE] [-n] [--chdir CHDIR] [--uid UID]
                        [--gid GID]
```

Start **and** monitor the agent process

optional arguments:

-h, --help show this help message **and** exit

--updates-drop-dir UPDATES_DROP_DIR
Where to look **for** agent updates

--agent-package-dir AGENT_PACKAGE_DIR
Path to the actual agent code

--pidfile PIDFILE The file to store the process **id** **in**. [default: **None**]

-n, --no-daemon If provided then do **not** run the process **in** the background.

--chdir CHDIR The directory to chdir to upon launch.

--uid UID The user **id** to run the supervisor **as**. *This setting **is** ignored on Windows.*

--gid GID The group **id** to run the supervisor **as**. *This setting **is** ignored on Windows.*

1.2 Development Commands

1.2.1 pyfarm-dev-fakerender

```
usage: pyfarm-dev-fakerender [-h] [--ram RAM] [--duration DURATION]
                             [--return-code RETURN_CODE]
                             [--duration-jitter DURATION_JITTER]
                             [--ram-jitter RAM_JITTER] -s START [-e END]
                             [-b BY] [--spew] [--segfault]
```

Very basic command line tool which vaguely simulates a render.

optional arguments:

-h, --help show this help message **and** exit

--ram RAM How much ram **in** megabytes the fake command should consume

--duration DURATION How many seconds it should take to run this command

```

--return-code RETURN_CODE
    The return code to return, declaring this flag
    multiple times will result in a random return code.
    [default: [0]]
--duration-jitter DURATION_JITTER
    Randomly add or subtract this amount to the total
    duration
--ram-jitter RAM_JITTER
    Randomly add or subtract this amount to the ram
-s START, --start START
    The start frame. If no other flags are provided this
    will also be the end frame.
-e END, --end END
    The end frame
-b BY, --by BY
    The by frame
--spew
    Spews lots of random output to stdout which is
    generally a decent stress test for log processing
    issues. Do note however that this will disable the
    code which is consuming extra CPU cycles. Also, use
    this option with care as it can generate several
    gigabytes of data per frame.
--segfault
    If provided then there's a 25% chance of causing a
    segmentation fault.

```

1.2.2 pyfarm-dev-fakework

```

usage: pyfarm-dev-fakework [-h] [--master-api MASTER_API]
                           [--agent-api AGENT_API] [--jobtype JOBTYP]
                           [--job JOB]

```

Quick **and** dirty script to create a job **type**, a job, **and** some tasks which are then posted directly to the agent. The primary purpose of this script **is** to test the internal of the job types

optional arguments:

```

-h, --help            show this help message and exit
--master-api MASTER_API
    The url to the master's api [default:
    http://127.0.0.1/api/v1]
--agent-api AGENT_API
    The url to the agent's api [default:
    http://127.0.0.1:50000/api/v1]
--jobtype JOBTYP     The job type to use [default: FakeRender]
--job JOB             If provided then this will be the job we pull tasks
    from and assign to the agent. Please note we'll only
    be pulling tasks that aren't running or assigned.

```

Environment Variables

PyFarm's agent has several environment variables which can be used to change the operation at runtime. For more information see the individual sections below.

PYFARM_JOBTYPE_ALLOW_CODE_EXECUTION_IN_MODULE_ROOT

If `True`, then function calls in the root of a job types's source code will result in an error when the work is assigned. By default, this value is set to `True`.

PYFARM_JOBTYPE_SUBCLASSES_BASE_CLASS

If `True` then job types which do not subclass from `pyfarm.jobtypes.core.jobtype.JobType` will raise an exception when work is assigned. By default, this value is set to `True`.

CHAPTER 3

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.

3.1 Agent

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

```
1  # The name of the render farm this agent belongs to.  If specified, the agent
2  # will not join a farm with a different name.  If not specified, the agent will
3  # join any farm.
4  farm_name: null
5
6  # The platform specific locations where the agent uuid
7  # file is default.  This can be overridden with --agent-id-file flag.
8  agent_id_file_platform_defaults:
9    linux: /etc/pyfarm/agent/uuid.dat
10   mac: /Library/pyfarm/agent/uuid.dat
11   bsd: /etc/pyfarm/agent/uuid.dat
12   windows: $LOCALAPPDATA/pyfarm/agent/uuid.dat
13
14  # The platform specific locations where the run control file is looked for
15  run_control_file_by_platform:
16    linux: /tmp/pyfarm/agent/should_be_running
17    mac: /tmp/pyfarm/agent/should_be_running
18    bsd: /tmp/pyfarm/agent/should_be_running
19    windows: $TEMP/pyfarm/agent/should_be_running
20
21  # The default location to store data.  $temp will expand to
22  # whatever pyfarm's data root is plus the application
23  # name (agent).  For example on Linux this would expand to
```

```
24 # /tmp/pyfarm/agent
25 agent_data_root: $temp
26
27 # Defines the number of seconds between iterations of pyfarm-supervisor's
28 # agent status check.
29 supervisor_interval: 5
30
31 # The location where the agent should change directories
32 # into upon starting. If this value is not set then no
33 # changes will be made.
34 agent_chdir:
35
36 # The location where static web files should be served from. This
37 # will default to using PyFarm's installation root.
38 agent_static_root: auto
39
40 # The default location where lock files should be stored. By
41 # default these will be stored alone side other data
42 # inside the `agent_data_root` value above.
43 lock_file_root: $agent_data_root/lock
44
45 # Locations of specific lock files
46 agent_lock_file: $lock_file_root/agent.pid
47 supervisor_lock_file: $lock_file_root/supervisor.pid
48
49 # Where user data for the agent is stored. ~ will be expanded
50 # to the current users's home directory.
51 agent_user_data: ~/.pyfarm/agent
52
53 # The default location where the agent should save logs to. This
54 # includes both logs from processes and the agent log itself.
55 agent_logs_root: $agent_data_root/logs
56
57 # The location where agent updates should be stored.
58 agent_updates_dir: $agent_data_root/updates
59
60 # The default port which the agent should use to serve the
61 # REST api.
62 agent_api_port: 50000
63
64 # The location where the the agent should save its own
65 # logging output to.
66 agent_log: $agent_logs_root/agent.log
67 supervisor_log: $agent_logs_root/supervisor.log
68
69 # Controls the log level for all loggers. Valid levels
70 # are 'debug', 'info', 'warning', 'error' and 'critical'.
71 agent_global_logger_level: info
72
73 # The user agent the master will use when connecting to the agent's
74 # REST api. This value should only be changed if the master's code
75 # is updated with a new user agent. Change this value has not effect
76 # on the master.
77 master_user_agent: PyFarm/1.0 (master)
78
79 # Configuration values which control how the url
80 # for the master is constructed. If 'master' is not set
81 # the --master flag will be required to start the agent.
```

```

82 master:
83 master_api_version: 1
84 master_api: http://$master/api/v$master_api_version
85
86 # The user agent the master uses to talke to the agent's
87 # REST api. This value should not be modified unless
88 # there's a specific reason to do so.
89 master_user_agent: PyFarm/1.0 (master)
90
91 # Controls how often the agent should reannounce itself
92 # to the master. The agent may be in contact with the master
93 # more often than this however during long period of
94 # inactivity this is how often the agent will 'inform' the
95 # master the agent is still online.
96 agent_master_reannounce: 120
97
98 # How many seconds the agent should spend attempting to inform
99 # the master that it's shutting down.
100 agent_shutdown_timeout: 15
101
102 # Number of seconds to offset from zero when calculating the
103 # http retry delay.
104 agent_http_retry_delay_offset: 1
105
106 # If an http request fails, use this as the base value
107 # to help determine how long we should wait before retrying. This
108 # value is then used to calculate the final delay:
109 # agent_http_retry_delay_offset + (random() * agent_http_retry_delay_factor)
110 agent_http_retry_delay_factor: 5
111
112 # Controls if the http client connection should be persistent or
113 # not. Generally this should always be True because the connection
114 # self-terminates after a short period of time anyway. For higher
115 # latency situations or with larger deployments this value should
116 # be False.
117 agent_http_persistent_connections: True
118
119 # When using persistent connections, twisted will sometimes run requests over
120 # connections that have already been closed by the server without realizing it.
121 # When this happens, we catch the resulting failure and retry the request.
122 # We do this up to $broken_connection_max_retry times, assuming that if
123 # it still fails at that point, there is probably something else wrong.
124 broken_connection_max_retry: 20
125
126 # If True then html templates will be reloaded with
127 # every request instead of cached.
128 agent_html_template_reload: False
129
130 # If True then reformat json output to be more human
131 # readable.
132 agent_pretty_json: True
133
134 # How often the agent should check for changes in ram. This value
135 # is used to ensure ram usage is checked at least this often though
136 # it may be checked more often due to other events (such as jobs
137 # running)
138 agent_ram_check_interval: 30
139

```

```
140 # If the ram has changed this may megabytes since the last
141 # check then report the change to the master.
142 agent_ram_report_delta: 100
143
144 # How much the agent should wait, in seconds, between
145 # each report about a change in ram.
146 agent_ram_max_report_frequency: 10
147
148 # The default network time server and version the agent
149 # should use to calculate its clock skew.
150 agent_ntp_server: pool.ntp.org
151 agent_ntp_server_version: 2
152
153 # The amount of time this agent is offset from what
154 # would be considered correct based on an atomic
155 # clock. If this value is set to auto the time will
156 # be calculated using NTP.
157 agent_time_offset: auto
158
159 # Physical and network information about the host the agent
160 # is running on. Setting these values to 'auto' will cause
161 # them to be initilized to the system's current
162 # configuration values.
163 agent_ram: auto
164 agent_cpus: auto
165 agent_hostname: auto
166
167 # When True this will enable a telnet connection
168 # to the agent which will present a Python interpreter
169 # upon connection. This is mainly used for debugging
170 # and direct manipulation of the agent. You can use
171 # the show() function once connected to see what
172 # objects are available.
173 agent_manhole: False
174 agent_manhole_port: 50001
175 agent_manhole_username: admin
176 agent_manhole_password: admin
177
178 # NOTE: The following values are used by the unittests and should be
179 # generally ignored for anything other than development.
180 agent_unittest:
181     client_redirect_target: http://example.com
182     client_api_test_url_https: https://httpbin.org
183     client_api_test_url_http: http://httpbin.org
184
185 # A list of paths or names where the `lspci` command can
186 # be called from on Linux. This is used to retrieve information
187 # about graphics cards installed on the system in
188 # `pyfarm.agent.sysinfo.graphics.graphics_cards`.
189 # If you need run the command with sudo you may also specify an entry
190 # like this:
191 # - sudo lspci
192 sysinfo_command_lspci:
193     - lspci
194     - /bin/lspci
195     - /sbin/lspci
196     - /usr/sbin/lspci
197     - /usr/bin/lspci
```

```

198 # If this is False, the agent will not allow two or more assignments to run
199 # on this node at the same time.
200 # If we still have at least one assignment with at least one task that isn't
201 # failed or done, new assignments will be rejected.
202 agent_allow_sharing: False
203

```

3.2 Job Types

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

```

1  # When set to True caching of job types will be enabled. When set to
2  # False caching is disabled and every job type will be retrieved from
3  # the master directly.
4  jobtype_enable_cache: True
5
6  # If True then output from all processes will be sent directly to
7  # the agent's logger(s) instead of to the log file associated
8  # with each process.
9  jobtype_capture_process_output: False
10
11 # The location where tasks should be logged
12 jobtype_task_logs: $agent_logs_root/tasks
13
14 # The filename to an individual log file. This filename supports several
15 # internal variables:
16 #
17 #   $YEAR - The current year
18 #   $MONTH - The current month
19 #   $DAY - The current day
20 #   $HOUR - The current hour
21 #   $MINUTE - The current minute
22 #   $JOB - The id of the job this log is for
23 #   $PROCESS - The uuid of the process object responsible for creating the log
24 #
25 # In addition to the above you can, as with any configuration variable,
26 # also use environment variables in the filename.
27 # Path separators ("/" and "\") are not allowed.
28 jobtype_task_log_filename:
29     $YEAR-$MONTH-$DAY-$HOUR-$MINUTE-$SECOND-$JOB-$PROCESS.csv
30
31 # store cached source code from the master. Note
32 # that $temp will be expanded to the local system's
33 # temp directory. If this directory does not exist
34 # it will be created. Leaving this value blank will
35 # disable job type caching.
36 jobtype_cache_directory: $temp/jobtype_cache
37
38 # The root directory that the default implementation of JobType.tempdir()
39 # will create a path using tempfile.mkdtemp.
40 jobtype_tempdir_root: $temp/tempdir/$JOBTYPE_UUID
41
42 # If True then expand environment variables in file paths.
43 jobtype_expandvars: True

```

```
44 # If True, then ignore any errors produced when trying
45 # to map users and groups to IDs. This will cause the
46 # underlying methods in the job type to instead run
47 # as the job type's owner instead, ignoring what the
48 # incoming job requests.
49 # NOTE: This value is not used on Windows.
50 jobtype_ignore_id_mapping_errors: False
51
52
53 # Any additional key/value pairs to include
54 # in the environment of a process launched
55 # by a job type.
56 jobtype_default_environment: {}
57
58 # If True then a job type's get_environment() method
59 # will also include the operating system's environment.
60 # The environment is constructed at the time the agent
61 # is launched, modifications to the agent's environment
62 # during execution are not included.
63 jobtype_include_os_environ: False
64
65 # Configures the thread pool used by job types
66 # for logging.
67 jobtype_logging_threadpool:
68     # Setting this value to something smaller than `1` will result
69     # in an exception being raised. This value also cannot be larger
70     # than `max_threads` below.
71     min_threads: 3
72
73     # This value must be greater than or equal to `min_threads`
74     # above. You may also set this value to 'auto' meaning the
75     # number of processors times 1.5 or 20 (whichever is lower).
76     max_threads: auto
77
78     # As log messages are sent from processes they are stored
79     # in an in memory queue. When the number of messages is higher
80     # than this number a thread will be spawned to consume the
81     # data and flush it into a file object.
82     max_queue_size: 10
83
84     # Most often the operating system will control how often data
85     # is written to disk from a file object. This value overrides
86     # that behavior and forces the file object to flush to disk
87     # after this many messages have been processed.
88     flush_lines: 100
```

4.1 Subpackages

4.1.1 pyfarm.agent.entrypoints package

Submodules

pyfarm.agent.entrypoints.development module

Development

Contains entry points which constructs the command line tools used for development such as `pyfarm-dev-fakerender` and `pyfarm-dev-fakework`.

`pyfarm.agent.entrypoints.development.fake_render()`

`pyfarm.agent.entrypoints.development.fake_work()`

`pyfarm.agent.entrypoints.development.random()` $\rightarrow x$ in the interval $[0, 1)$.

pyfarm.agent.entrypoints.main module

Main

The main module which constructs the entrypoint for the `pyfarm-agent` command line tool.

class `pyfarm.agent.entrypoints.main.AgentEntryPoint`

Bases: `object`

Main object for parsing command line options

agent_api

```
start ()
stop ()
status ()
```

pyfarm.agent.entrypoints.parser module

Parser

Module which forms the basis of a custom `argparse` based command line parser which handles setting configuration values automatically.

```
pyfarm.agent.entrypoints.parser.assert_parser (func)
    ensures that the instance argument passed along to the validation function contains data we expect
```

```
pyfarm.agent.entrypoints.parser.ip (*args, **kwargs)
    make sure the ip address provided is valid
```

```
pyfarm.agent.entrypoints.parser.port (*args, **kwargs)
    convert and check to make sure the provided port is valid
```

```
pyfarm.agent.entrypoints.parser.uuid_type (*args, **kwargs)
    validates that a string is a valid UUID type
```

```
pyfarm.agent.entrypoints.parser.uidgid (*args, **kwargs)
    Retrieves and validates the user or group id for a command line flag
```

```
pyfarm.agent.entrypoints.parser.direxists (*args, **kwargs)
    checks to make sure the directory exists
```

```
pyfarm.agent.entrypoints.parser.fileexists (*args, **kwargs)
    checks to make sure the provided file exists
```

```
pyfarm.agent.entrypoints.parser.number (*args, **kwargs)
    convert the given value to a number
```

```
pyfarm.agent.entrypoints.parser.enum (*args, **kwargs)
    ensures that value is a valid entry in enum
```

```
class pyfarm.agent.entrypoints.parser.ActionMixin (*args, **kwargs)
    Bases: object
```

A mixin which overrides the `__init__` and `__call__` methods on an action so we can:

- Setup attributes to manipulate the config object when the arguments are parsed
- Ensure we all required arguments are present
- Convert the `type` keyword into an internal representation so we don't require as much work when we add arguments to the parser

```
TYPE_MAPPING = {<function isdir>: <function direxists>, <type 'int'>: <functools.partial>
```

```
pyfarm.agent.entrypoints.parser.mix_action (class_)
```

```
pyfarm.agent.entrypoints.parser.StoreAction
    alias of _StoreAction
```

```
pyfarm.agent.entrypoints.parser.SubParsersAction
    alias of _SubParsersAction
```

`pyfarm.agent.entrypoints.parser.StoreConstAction`
alias of `_StoreConstAction`

`pyfarm.agent.entrypoints.parser.StoreTrueAction`
alias of `_StoreTrueAction`

`pyfarm.agent.entrypoints.parser.StoreFalseAction`
alias of `_StoreFalseAction`

`pyfarm.agent.entrypoints.parser.AppendAction`
alias of `_AppendAction`

`pyfarm.agent.entrypoints.parser.AppendConstAction`
alias of `_AppendConstAction`

class `pyfarm.agent.entrypoints.parser.AgentArgumentParser` (*args, **kwargs)
Bases: `argparse.ArgumentParser`

A modified `ArgumentParser` which interfaces with the agent's configuration.

pyfarm.agent.entrypoints.supervisor module

`pyfarm.agent.entrypoints.supervisor.supervisor()`

pyfarm.agent.entrypoints.utility module

Utility

Small objects and functions which facilitate operations on the main entry point class.

`pyfarm.agent.entrypoints.utility.start_daemon_posix(log, chdir, uid, gid)`

Runs the agent process via a double fork. This basically a duplicate of Marcechal's original code with some adjustments:

http://www.jejik.com/articles/2007/02/a_simple_unix_linux_daemon_in_python/

Source files from his post are here: <http://www.jejik.com/files/examples/daemon.py> <http://www.jejik.com/files/examples/daemon3x.py>

Module contents

Entry Points

This module contains several subpackages which serve as the basis for the command line tools for the agent.

4.1.2 pyfarm.agent.http package

Subpackages

pyfarm.agent.http.api package

Submodules

pyfarm.agent.http.api.assign module

```
class pyfarm.agent.http.api.assign.Assign(agent)
    Bases: pyfarm.agent.http.api.base.APIResource

    isLeaf = False

    SCHEMAS = {'POST': <Schema({'job': <Schema({'cpus': Any([<type 'int'>, <type 'long'>
    post (**kwargs)
```

pyfarm.agent.http.api.base module

Base

Contains the base resources used for building up the root of the agent's api.

```
class pyfarm.agent.http.api.base.APIResource
    Bases: pyfarm.agent.http.core.resource.Resource

    Base class for all api resources

    isLeaf = True

    ALLOWED_CONTENT_TYPE = frozenset(['application/json', None])
    DEFAULT_CONTENT_TYPE = frozenset(['application/json'])
    ALLOWED_ACCEPT = frozenset(['*/*', 'application/json', None])
    DEFAULT_ACCEPT = frozenset(['application/json'])
```

```
class pyfarm.agent.http.api.base.APIRoot
    Bases: pyfarm.agent.http.api.base.APIResource

    isLeaf = False
```

```
class pyfarm.agent.http.api.base.Versions
    Bases: pyfarm.agent.http.api.base.APIResource
```

Returns a list of api versions which this agent will support

GET /api/v1/versions/ HTTP/1.1
Request

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

Response

```

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

{
    "versions": [1]
}

```

```

isLeaf = True
get (**kwargs)

```

pyfarm.agent.http.api.config module

Config

Contains the endpoint for viewing and working with the configuration on the agent.

```

class pyfarm.agent.http.api.config.Config
    Bases: pyfarm.agent.http.api.base.APIResource
    isLeaf = False
    get (**kwargs)

```

pyfarm.agent.http.api.software module

Check Software Endpoint

This endpoint is used to instruct the agent to check whether a given version of a software is installed and usable locally.

```

class pyfarm.agent.http.api.software.CheckSoftware
    Bases: pyfarm.agent.http.api.base.APIResource
    Requests the agent to check whether a given software version is installed locally.
    The agent will asynchronously update its software information on the master.

```

```

POST /api/v1/check_software HTTP/1.1
Request

```

```

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

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

```

Response

```

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

```

```

SCHEMAS = {'POST': <Schema({'version': Any([<type 'str'>, <type 'unicode'>])}, 'softwa
isLeaf = False

```

```
testing = False
post (**kwargs)
```

pyfarm.agent.http.api.state module

```
class pyfarm.agent.http.api.state.Stop
    Bases: pyfarm.agent.http.api.base.APIResource
    isLeaf = False
    SCHEMAS = {'POST': <Schema({'wait': <type 'bool'>}, extra=PREVENT_EXTRA, required=False)>
    post (**kwargs)

class pyfarm.agent.http.api.state.Restart
    Bases: pyfarm.agent.http.api.base.APIResource
    isLeaf = False
    post (**kwargs)

class pyfarm.agent.http.api.state.Status
    Bases: pyfarm.agent.http.api.base.APIResource
    isLeaf = False
    get (**_)
```

pyfarm.agent.http.api.tasklogs module

```
class pyfarm.agent.http.api.tasklogs.TaskLogs
    Bases: pyfarm.agent.http.api.base.APIResource
    get (**kwargs)
```

Get the contents of the specified task log. The log will be returned in CSV format with the following fields:

- ISO8601 timestamp
- Stream number (0 == stdout, 1 == stderr)
- Line number
- Parent process ID
- Message the process produced

The log file identifier is configurable and relies on the *jobtype_task_log_filename* configuration option. See the configuration documentation for more information about the default value.

GET /api/v1/tasklogs/<identifier> HTTP/1.1

Request

```
GET /api/v1/tasklogs/<identifier> HTTP/1.1
```

Response

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

2015-05-07T23:42:53.730975,0,15,42,Hello world
```

Statuscode 200 The log file was found, it's content will be returned.

pyfarm.agent.http.api.tasks module

class pyfarm.agent.http.api.tasks.**Tasks**

Bases: *pyfarm.agent.http.api.base.APIResource*

get (***kwargs*)

Returns all tasks which are currently being processed locally by the agent.

GET /api/v1/tasks/ HTTP/1.1

Request

```
GET /api/v1/tasks/ HTTP/1.1
```

Response

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

[{"id": "732c1ef0-9488-4914-adeb-c29f481f3f5b",
  "frame": 1,
  "attempt": 1
},
{
  "id": "34ce3964-b654-4ad4-8416-f5ddba67806e",
  "frame": 2,
  "attempt": 1
}]
```

Statuscode 200 The request was processed successfully

delete (***kwargs*)

HTTP endpoint for stopping and deleting an individual task from this agent.

Warning: If the specified task is part of a multi-task assignment, all tasks in this assignment will be stopped, not just the specified one.

This will try to asynchronously stop the assignment by killing all its child processes. If that isn't successful, this will have no effect.

DELETE /api/v1/tasks/<int:task_id> HTTP/1.1

Request

```
DELETE /api/v1/tasks/1 HTTP/1.1
```

Response

```
HTTP/1.1 202 ACCEPTED
Content-Type: application/json
```

Statuscode 202 The task was found and will be stopped.

Statuscode 204 Nothing to do, no task matching the request was found

Statuscode 400 There was a problem with the request, check the error

pyfarm.agent.http.api.update module

Update Endpoint

This endpoint is used to instruct the agent to download and apply an update.

class pyfarm.agent.http.api.update.**Update**

Bases: *pyfarm.agent.http.api.base.APIResource*

Requests the agent to download and apply the specified version of itself. Will make the agent restart at the next opportunity.

POST /api/v1/update **HTTP/1.1**

Request

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

{
  "version": 1.2.3
}
```

Response

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

```
SCHEMAS = {'POST': <Schema({'version': Any([<type 'str'>, <type 'unicode'>])}), extra=
isLeaf = False
post (**kwargs)
```

Module contents

API

This package contains the components that form the agent's api. The objects contained in this section act as an interface layer between an incoming http request and the agent's internals.

pyfarm.agent.http.core package

Submodules

pyfarm.agent.http.core.client module

HTTP Client

The client library the manager uses to communicate with the master server.

class `pyfarm.agent.http.core.client.HTTPLog`

Bases: `object`

Provides a wrapper around the http logger so requests and responses can be logged in a standardized fashion.

static `queue (method, url, uid=None)`

Logs the request we're asking treq to queue

static `response (response, uid=None)`

Logs the return code of a request that treq completed

static `error (failure, uid=None, method=None, url=None)`

Called when the treq request experiences an error and calls the `errback` method.

`pyfarm.agent.http.core.client.build_url (url, params=None)`

Builds the full url when provided the base url and some url parameters:

```
>>> build_url("/foobar", {"first": "foo", "second": "bar"})
'/foobar?first=foo&second=bar'
>>> build_url("/foobar bar/")
'/foobar%20bar/'
```

Parameters

- **url** (*str*) – The url to build off of.
- **params** (*dict*) – A dictionary of parameters that should be added on to url. If this value is not provided url will be returned by itself. Arguments to a url are unordered by default however they will be sorted alphabetically so the results are repeatable from call to call.

`pyfarm.agent.http.core.client.http_retry_delay (offset=None, factor=None, rand=None)`

Returns a floating point value that can be used to delay an http request. The main purpose of this is to ensure that not all requests are run with the same interval between them.

The basic formula for the retry delay is:

```
offset * (random() * factor)
```

Parameters

- **factor** (*int* or *float*) – The factor to multiply the output from `random()` by. This defaults to the `agent_http_retry_delay_factor` configuration variable.
- **offset** – The initial offset to start the calculation at. This defaults to the `agent_http_retry_delay_offset` configuration variable.
- **rand** – A callable to determine randomness, defaulting to `random()`. This is mainly used for testing purposes.

class `pyfarm.agent.http.core.client.Request`

Bases: `pyfarm.agent.http.core.client.Request`

Contains all the information used to perform a request such as the `method`, `url`, and original keyword arguments (`kwargs`). These values contain the basic information necessary in order to `retry()` a request.

retry (***kwargs*)

When called this will rerun the original request with all of the original arguments to `request()`

Parameters **kwargs** – Additional keyword arguments which should override the original keyword argument(s).

class `pyfarm.agent.http.core.client.Response` (*deferred, response, request*)

Bases: `twisted.internet.protocol.Protocol`

This class receives the incoming response body from a request constructs some convenience methods and attributes around the data.

Parameters

- **deferred** (*Deferred*) – The deferred object which contains the target callback and errback.
- **response** – The initial response object which will be passed along to the target deferred.
- **request** (*Request*) – Named tuple object containing the method name, url, headers, and data.

data ()

Returns the data currently contained in the buffer.

Raises **RuntimeError** – Raised if this method is called before all data has been received.

json (*loader=<function loads>*)

Returns the json data from the incoming request

Raises

- **RuntimeError** – Raised if this method is called before all data has been received.
- **ValueError** – Raised if the content type for this request is not application/json.

dataReceived (*data*)

Overrides `Protocol.dataReceived()` and appends data to `_body`.

connectionLost (*reason=<twisted.python.failure.Failure twisted.internet.error.ConnectionDone: Connection was closed cleanly.>*)

Overrides `Protocol.connectionLost()` and sets the `_done` when complete. When called with `ResponseDone` for reason this method will call the callback on `_deferred`

`pyfarm.agent.http.core.client.request` (*method, url, **kwargs*)

Wrapper around `treq.request()` with some added arguments and validation.

Parameters

- **method** (*str*) – The HTTP method to use when making the request.
- **url** (*str*) – The url this request will be made to.
- **data** (*str, list, tuple, set, dict*) – The data to send along with some types of requests such as POST or PUT
- **headers** (*dict*) – The headers to send along with the request to url. Currently only single values per header are supported.
- **callback** (*function*) – The function to deliver an instance of *Response* once we receive and unpack a response.
- **errback** (*function*) – The function to deliver an error message to. By default this will use `log.err()`.
- **response_class** (*class*) – The class to use to unpack the internal response. This is mainly used by the unittests but could be used elsewhere to add some custom behavior to the unpack process for the incoming response.

Raises **NotImplementedError** – Raised whenever a request is made of this function that we can't implement such as an invalid http scheme, request method or a problem constructing data to an api.

`pyfarm.agent.http.core.client.random()` → x in the interval $[0, 1)$.

pyfarm.agent.http.core.resource module

Resource

Base resources which can be used to build top leve documents, pages, or other types of data for the web.

class `pyfarm.agent.http.core.resource.Resource`

Bases: `twisted.web.resource.Resource`

Basic subclass of `_Resource` for passing requests to specific methods. Unlike `_Resource` however this will also handle:

- Templates
- Content type discovery and validation
- Handling of deferred responses
- Validation of POST/PUT data against a schema

Variables

- **TEMPLATE** (*string*) – The name of the template this class will use when rendering an html view.
- **SCHEMAS** – A dictionary of schemas to validate the data of an incoming request against. The structure of this dictionary is:

```
{http method: <instance of voluptuous.Schema>}
```

If the schema validation fails the request will be rejected with 400 BAD REQUEST.

- **ALLOWED_CONTENT_TYPE** – An instance of `frozenset` which describes what this resource is going to allow in the Content-Type header. The request and this instance must share at least on entry in common. If not, the request will be rejected with 415 UNSUPPORTED MEDIA TYPE.

This must be defined in subclass

- **ALLOWED_ACCEPT** – An instance of `frozenset` which describes what this resource is going to allow in the Accept header. The request and this instance must share at least one entry in common. If not, the request will be rejected with 406 NOT ACCEPTABLE.

This must be defined in subclass

- **DEFAULT_ACCEPT** – If Accept header is not present in the request, use this as the value instead. This defaults to `frozenset(["*/*"])`
- **DEFAULT_CONTENT_TYPE** – If Content-Type header is not present in the request, use this as the value instead. This defaults to `frozenset([""])`

TEMPLATE = `NotImplemented`

SCHEMAS = `{}`

```
ALLOWED_ACCEPT = NotImplemented
ALLOWED_CONTENT_TYPE = NotImplemented
DEFAULT_ACCEPT = frozenset(['*/'])
DEFAULT_CONTENT_TYPE = frozenset([None])

template
    Loads the template provided but the partial path in TEMPLATE on the class.

methods ()
    Returns a tuple of methods which an instance of this class implements

get_content_type (request)
    Return the Content-Type header(s) in the request or DEFAULT_CONTENT_TYPE if the header is not
    set.

get_accept (request)
    Return the Accept header(s) in the request or DEFAULT_ACCEPT if the header is not set.

putChild (path, child)
    Overrides the builtin putChild() so we can return the results for each call and use them externally.

error (request, code, message)
    Writes the proper out an error response message depending on the content type in the request

set_response_code_if_not_set (request, code)
    Sets the response code if one has not already been set

render_tuple (request, response)
    Takes a response tuple of (body, code, headers) or (body, code) and renders the resulting
    data onto the request.

render_deferred (*args, **kwargs)
    An inline callback used to unpack a deferred response object.

render (request)
```

pyfarm.agent.http.core.server module

HTTP Server

HTTP server responsible for serving requests that control or query the running agent. This file produces a service that the `pyfarm.agent.manager.service.ManagerServiceMaker` class can consume on start.

```
class pyfarm.agent.http.core.server.Site (resource, requestFactory=None, *args,
                                          **kwargs)
    Bases: twisted.web.server.Site

    Site object similar to Twisted's except it also carries along some of the internal agent data.

    displayTracebacks = True

class pyfarm.agent.http.core.server.StaticPath (*args, **kwargs)
    Bases: twisted.web.static.File

    More secure version of File that does not list directories. In addition this will also sending along a response
    header asking clients to cache to data.

    EXPIRES = 604800

    ALLOW_DIRECTORY_LISTING = False
```

render (*request*)
Overrides `File.render()` and sets the expires header

directoryListing ()
Override which ensures directories cannot be listed

pyfarm.agent.http.core.template module

Template

Interface methods for working with the Jinja template engine.

class `pyfarm.agent.http.core.template.InMemoryCache`

Bases: `jinja2.bccache.BytecodeCache`

Caches Jinja templates into memory after they have been loaded and compiled.

cache = {}

clear ()

load_bytecode (*bucket*)

dump_bytecode (*bucket*)

class `pyfarm.agent.http.core.template.Environment` (***kwargs*)

Bases: `jinja2.environment.Environment`

Implementation of `jinja2.Environment` class which reads from our configuration object and establishes the default functions we can use in a template.

Module contents

Core

This module contains the core libraries necessary for working with HTTP requests and responses.

Submodules

pyfarm.agent.http.system module

`pyfarm.agent.http.system.mb` (*value*)

`pyfarm.agent.http.system.seconds` (*value*)

class `pyfarm.agent.http.system.HTMLResource`

Bases: `pyfarm.agent.http.core.resource.Resource`

ALLOWED_CONTENT_TYPE = `frozenset(['', 'text/html'])`

ALLOWED_ACCEPT = `frozenset(['*/*', 'text/html'])`

class `pyfarm.agent.http.system.Index`

Bases: `pyfarm.agent.http.system.HTMLResource`

serves request for the root, '/', target

TEMPLATE = `'index.html'`

```
    get (**_)

class pyfarm.agent.http.system.Configuration
    Bases: pyfarm.agent.http.system.HTMLResource

    TEMPLATE = 'configuration.html'
    HIDDEN_FIELDS = ('agent', 'agent_pretty_json')
    EDITABLE_FIELDS = ('agent_cpus', 'agent_hostname', 'master_api', 'master', 'agent_ram_
    get (**_)
```

Module contents

HTTP Components

This sub-module contains all the code necessary to interact via HTTP from both a client and a server perspective.

4.1.3 pyfarm.agent.sysinfo package

Submodules

pyfarm.agent.sysinfo.cpu module

CPU

Contains information about the cpu and its relation to the operating system such as load, processing times, etc.

`pyfarm.agent.sysinfo.cpu.cpu_name()`

Returns the full name of the CPU installed in the system.

`pyfarm.agent.sysinfo.cpu.total_cpus(logical=True)`

Returns the total number of cpus installed on the system.

Parameters `logical (bool)` – If True the return the number of cores the system has. Setting this value to False will instead return the number of physical cpus present on the system.

`pyfarm.agent.sysinfo.cpu.load(interval=1)`

Returns the load across all cpus value from zero to one. A value of 1.0 means the average load across all cpus is 100%.

`pyfarm.agent.sysinfo.cpu.user_time()`

Returns the amount of time spent by the cpu in user space

`pyfarm.agent.sysinfo.cpu.system_time()`

Returns the amount of time spent by the cpu in system space

`pyfarm.agent.sysinfo.cpu.idle_time()`

Returns the amount of time spent by the cpu in idle space

`pyfarm.agent.sysinfo.cpu.iowait()`

Returns the amount of time spent by the cpu waiting on io

Note: on platforms other than linux this will return None

pyfarm.agent.sysinfo.disks module

Disks

Contains information about the local disks.

class pyfarm.agent.sysinfo.disks.**DiskInfo** (*mountpoint, free, size*)
Bases: `tuple`

free
Alias for field number 1

mountpoint
Alias for field number 0

size
Alias for field number 2

pyfarm.agent.sysinfo.disks.**disks** (*as_dict=False*)
Returns a list of disks in the system, in the form of *DiskInfo* objects.

Parameters *as_dict* (*bool*) – If True then return a dictionary value instead of *DiskInfo* instances. This is mainly used by the agent to eliminate an extra loop for translation.

pyfarm.agent.sysinfo.graphics module

Graphics

Contains information about the installed graphics cards

exception pyfarm.agent.sysinfo.graphics.**GPULookupError** (*value*)
Bases: `exceptions.Exception`

pyfarm.agent.sysinfo.graphics.**graphics_cards**()
Returns a list of the full names of GPUs installed in this system

pyfarm.agent.sysinfo.memory module

Memory

Provides information about memory including swap usage, system memory usage, and general capacity information.

pyfarm.agent.sysinfo.memory.**used_ram**()
Amount of physical memory currently in use by applications

pyfarm.agent.sysinfo.memory.**free_ram**()
Amount of physical memory free for application use

pyfarm.agent.sysinfo.memory.**total_ram**()
Total physical memory installed on the system

pyfarm.agent.sysinfo.memory.**process_memory**()
Total amount of memory in use by this process

pyfarm.agent.sysinfo.memory.**total_consumption**()
Total amount of memory consumed by this process and any child process spawned by the parent process. This includes any grandchild processes.

pyfarm.agent.sysinfo.network module

Network

Returns information about the network including ip address, dns, data sent/received, and some error information.

const IP_PRIVATE set of private class A, B, and C network ranges

See also:

RFC 1918

const IP_NONNETWORK set of non-network address ranges including all of the above constants except the IP_PRIVATE

`pyfarm.agent.sysinfo.network.mac_addresses (long_addresses=False, as_integers=False)`

Returns a tuple of all mac addresses on the system.

Parameters

- **long_addresses** (*bool*) – Some adapters will specify a mac address which is longer than the standard value of six pairs. Setting this value to True will allow these to be displayed.
- **as_integers** (*bool*) – When True convert all mac addresses to integers.

`pyfarm.agent.sysinfo.network.hostname (trust_name_from_ips=True)`

Returns the hostname which the agent should send to the master.

Parameters **trust_resolved_name** (*bool*) – If True and all addresses provided by `addresses()` resolve to a single hostname then just return that name as it's the most likely hostname to be accessible by the rest of the network.

`pyfarm.agent.sysinfo.network.addresses (private_only=True)`

Returns a tuple of all non-local ip addresses.

`pyfarm.agent.sysinfo.network.interfaces ()`

Returns the names of all valid network interface names

pyfarm.agent.sysinfo.software module

Software

Contains utilities to check the availability of certain software on the local machine.

exception `pyfarm.agent.sysinfo.software.VersionNotFound`

Bases: `exceptions.Exception`

`pyfarm.agent.sysinfo.software.get_software_version_data (*args, **kwargs)`

Asynchronously fetches the known data about the given software version from the master.

Parameters

- **software** (*str*) – The name of the software to get data for
- **version** (*str*) – The name of the version to get data for

Returns Returns information about the given software version from the master

`pyfarm.agent.sysinfo.software.get_discovery_code (*args, **kwargs)`

Asynchronously fetches the discovery code for the given software version from the master.

Parameters

- **software** (*str*) – The name of the software to get the discovery code for
- **version** (*str*) – The name of the version to get the discovery code for

Returns Returns the discovery code from the master/

`pyfarm.agent.sysinfo.software.check_software_availability(*args, **kwargs)`

Asynchronously checks for the availability of a given software in a given version. Will pass True to its callback function if the software could be found, False otherwise. Works only for software versions that have a discovery registered on the master.

Parameters

- **software** (*str*) – The name of the software to check for
- **version** (*str*) – The name of the version to check for

pyfarm.agent.sysinfo.system module**System**

Information about the operating system including type, filesystem information, and other relevant information. This module may also contain os specific information such as the Linux distribution, Windows version, bitness, etc.

`pyfarm.agent.sysinfo.system.filesystem_is_case_sensitive()`
returns True if the file system is case sensitive

`pyfarm.agent.sysinfo.system.environment_is_case_sensitive()`
returns True if the environment is case sensitive

`pyfarm.agent.sysinfo.system.machine_architecture(arch='x86_64')`
returns the architecture of the host itself

`pyfarm.agent.sysinfo.system.interpreter_architecture()`
returns the architecture of the interpreter itself (32 or 64)

`pyfarm.agent.sysinfo.system.uptime()`
Returns the amount of time the system has been running in seconds.

`pyfarm.agent.sysinfo.system.operating_system(plat='linux2')`
Returns the operating system for the given platform. Please note that while you can call this function directly you're more likely better off using values in `pyfarm.core.enums` instead.

pyfarm.agent.sysinfo.user module**User**

Returns information about the current user such as the user name, admin access, or other related information.

`pyfarm.agent.sysinfo.user.username()`
Returns the current user name using the most native api we can import. On Linux for example this will use the `pwd` module but on Windows we try to use `win32api`.

`pyfarm.agent.sysinfo.user.is_administrator()`
Return True if the current user is root (Linux) or running as an Administrator (Windows).

Module contents

Top level module which provides information about the operating system, system memory, network, and processor related information

4.2 Submodules

4.2.1 pyfarm.agent.config module

Configuration

Central module for storing and working with a live configuration objects. This module instances *ConfigurationWithCallbacks* onto `config`. Attempting to reload this module will not reinstance the `config` object.

The `config` object should be directly imported from this module to be used:

```
>>> from pyfarm.agent.config import config
```

```
class pyfarm.agent.config.LoggingConfiguration (data=None, environment=None,
                                              load=True)
```

Bases: `pyfarm.core.config.Configuration`

Special configuration object which logs when a key is changed in a dictionary. If the reactor is not running then log messages will be queued until they can be emitted so they are not lost.

`__expandvars` (*value*)

Performs variable expansion for *value*. This method is run when a string value is returned from `get()` or `__getitem__()`. The default behavior of this method is to recursively expand variables using sources in the following order:

- The environment, `os.environ`
- The environment (from the configuration), `env`
- Other values in the configuration
- ~ to the user's home directory

For example, the following configuration:

```
foo: foo
bar: bar
foobar: $foo/$bar
path: ~/$foobar/$TEST
```

Would result in the following assuming `$TEST` is an environment variable set to `somevalue` and the current user's name is `user`:

```
{
  "foo": "foo",
  "bar": "bar",
  "foobar": "foo/bar",
  "path": "/home/user/foo/bar/somevalue"
}
```

`MODIFIED` = `'modified'`

CREATED = 'created'

DELETED = 'deleted'

pop (*key*, **args*)

Deletes the provided key and triggers a delete event using *changed()*.

clear ()

Deletes all keys in this object and triggers a delete event using *changed()* for each one.

update (*data=None*, ***kwargs*)

Updates the data held within this object and triggers the appropriate events with *changed()*.

changed (*change_type*, *key*, *new_value=<object object>*, *old_value=<object object>*)

This method is run whenever one of the keys in this object changes.

master_contacted (*update=True*, *announcement=False*)

Simple method that will update the *last_master_contact* and then return the result.

Parameters *update* (*bool*) – Setting this value to False will just return the current value instead of updating the value too.

class *pyfarm.agent.config.ConfigurationWithCallbacks* (*data=None*, *environment=None*, *load=True*)

Bases: *pyfarm.agent.config.LoggingConfiguration*

Subclass of *LoggingDictionary* that provides the ability to run a function when a value is changed.

callbacks = {}

classmethod *register_callback* (*key*, *callback*, *append=False*)

Register a function as a callback for key. When key is set the given callback will be run by *changed()*

Parameters

- **key** (*string*) – the key which when changed in any way will execute callback
- **callback** (*callable*) – the function or method to register
- **append** (*boolean*) – by default attempting to register a callback which has already been registered will do nothing, setting this to True overrides this behavior.

classmethod *deregister_callback* (*key*, *callback*)

Removes any callback(s) that are registered with the provided key

clear (*callbacks=False*)

Performs the same operations as *dict.clear()* except this method can also clear any registered callbacks if requested.

changed (*change_type*, *key*, *new_value=<object object>*, *old_value=<object object>*)

This method is called internally whenever a given key changes which in turn will pass off the change to any registered callback(s).

pyfarm.agent.config.configure_logger_level ()

When called this will set the root logger level based on the *agent_global_logger_level* configuration variable.

4.2.2 pyfarm.agent.manhole module

Manhole

Provides a way to access the internals of the agent via the telnet protocol.

class pyfarm.agent.manhole.**LoggingManhole** (*namespace=None*)

Bases: twisted.conch.manhole.ColoredManhole

A slightly modified implementation of ColoredManhole which logs information to the logger so we can track activity in the agent's log.

connectionMade ()

connectionLost (*reason*)

lineReceived (*line*)

class pyfarm.agent.manhole.**TransportProtocolFactory** (*portal*)

Bases: object

Glues together a portal along with the TelnetTransport and AuthenticatingTelnetProtocol objects. This class is instantiated onto the protocol attribute of the ServerFactory class in build_manhole().

class pyfarm.agent.manhole.**TelnetRealm**

Bases: object

Wraps together ITelnetProtocol, TelnetBootstrapProtocol, ServerProtocol and ColoredManhole in *requestAvatar()* which will provide the interface to the manhole.

NAMESPACE = None

requestAvatar (_, *interfaces)

pyfarm.agent.manhole.**show** (*x=<object object>*)

Display the data attributes of an object in a readable format

pyfarm.agent.manhole.**manhole_factory** (*namespace, username, password*)

Produces a factory object which can be used to listen for telnet connections to the manhole.

4.2.3 pyfarm.agent.service module

Manager Service

Sends and receives information from the master and performs systems level tasks such as log reading, system information gathering, and management of processes.

class pyfarm.agent.service.**Agent**

Bases: object

Main class associated with getting the internals of the agent's operations up and running including adding or updating itself with the master, starting the periodic task manager, and handling shutdown conditions.

classmethod **agent_api** ()

Return the API url for this agent or None if *agent_id* has not been set

classmethod **agents_endpoint** ()

Returns the API endpoint for used for updating or creating agents on the master

shutting_down

repeating_call (*delay, function, function_args=None, function_kwargs=None, now=True, repeat_max=None, function_id=None*)

Causes function to be called repeatedly up until repeat_max or until stopped.

Parameters

- **delay** (*int*) – Number of seconds to delay between calls of function.

Note: `delay` is an approximate interval between when one call ends and the next one begins. The exact time can vary due to how the Twisted reactor runs, how long it takes function to run and what else may be going on in the agent at the time.

- **function** – A callable function to run
- **function_args** (*tuple*, *list*) – Arguments to pass into function
- **function_kwargs** (*dict*) – Keywords to pass into function
- **now** (*bool*) – If True then run function right now in addition to scheduling it.
- **repeat_max** (*int*) – Repeat calling function this many times. If not provided then we'll continue to repeat calling function until the agent shuts down.
- **function_id** (*uuid.UUID*) – Used internally to track a function's execution count. This keyword exists so if you call `repeating_call()` multiple times on the same function or method it will handle `repeat_max` properly.

should_reannounce ()

Small method which acts as a trigger for `reannounce()`

reannounce (*args, **kwargs)

Method which is used to periodically contact the master. This method is generally called as part of a scheduled task.

system_data (*reqquery_timeoffset=False*)

Returns a dictionary of data containing information about the agent. This is the information that is also passed along to the master.

build_http_resource ()

start (*shutdown_events=True*, *http_server=True*)

Internal code which starts the agent, registers it with the master, and performs the other steps necessary to get things running.

Parameters

- **shutdown_events** (*bool*) – If True register all shutdown events so certain actions, such as information the master we're going offline, can take place.
- **http_server** (*bool*) – If True then construct and serve the externally facing http server.

stop (*args, **kwargs)

Internal code which stops the agent. This will terminate any running processes, inform the master of the terminated tasks, update the state of the agent on the master.

sigint_handler (*)

post_shutdown_to_master (*args, **kwargs)

This method is called before the reactor shuts down and lets the master know that the agent's state is now offline

post_agent_to_master (*args, **kwargs)

Runs the POST request to contact the master. Running this method multiple times should be considered safe but is generally something that should be avoided.

callback_agent_id_set (*change_type*, *key*, *new_value*, *old_value*, *shutdown_events=True*)

When `agent_id` is created we need to:

- Register a shutdown event so that when the agent is told to shutdown it will notify the master of a state change.
- Star the scheduled task manager

callback_shutting_down_changed (*change_type, key, new_value, old_value*)

When *shutting_down* is changed in the configuration, set or reset *self.shutdown_timeout*

4.2.4 pyfarm.agent.testutil module

class `pyfarm.agent.testutil.skipIf` (*should_skip, reason*)

Bases: `object`

Wrapping a test with this class will allow the test to be skipped if *should_skip* evals as `True`.

`pyfarm.agent.testutil.random_port` (*bind='127.0.0.1'*)

Returns a random port which is not in use

`pyfarm.agent.testutil.create_jobtype` (*classname=None, sourcecode=None*)

Creates a job type on the master and fires a deferred when finished

class `pyfarm.agent.testutil.FakeRequestHeaders` (*test, headers*)

Bases: `object`

getRawHeaders (*header*)

class `pyfarm.agent.testutil.FakeAgent` (*stopped=None*)

Bases: `object`

stop ()

class `pyfarm.agent.testutil.ErrorCapturingParser` (**args, **kwargs*)

Bases: `pyfarm.agent.entrypoints.parser.AgentArgumentParser`

error (*message*)

class `pyfarm.agent.testutil.APITestServerResource`

Bases: `twisted.web.resource.Resource`

isLeaf = `False`

putChild (*path, child*)

handle (*request*)

render_POST (*request*)

render_PUT (*request*)

render_GET (*request*)

render_DELETE (*request*)

class `pyfarm.agent.testutil.APITestServer` (*url, code=None, response=None, headers=None*)

Bases: `object`

A object used for setting up a fake HTTP server which can respond to requests during a test.

class `pyfarm.agent.testutil.DummyRequest` (*postpath='/', session=None*)

Bases: `twisted.web.test.requesthelper.DummyRequest`

code = `200`

```

set_content (content)
    Sets the content of the request

setHeader (name, value)
    Default override, _DummyRequest.setHeader does not actually set the response headers. Instead it sets
    the value in a different location that's never used in an actual request.

getHeader (key)
    Default override, _DummyRequest.getHeader does something different than the real request object.

write (data)
    Default override, _DummyRequest.write asserts that data must be a bytes instance. In the real Re-
    quest.write implementation no such assertion is made.

class pyfarm.agent.testutil.TestCaseLogHandler (level=10)
    Bases: logging.Handler

    handle (record)

class pyfarm.agent.testutil.TestCase (methodName='runTest')
    Bases: twisted.trial._async_test.TestCase

    longMessage = True

    POP_CONFIG_KEYS = []

    RAND_LENGTH = 8

    maxDiff = None

    timeout = 15

    assertRaises (excClass, callableObj=None, *args, **kwargs)

    assertRaisesRegexp (expected_exception, expected_regexp, callable_obj=None, *args, **kwargs)

    assertDateAlmostEqual (date1, date2, second_deviation=0, microsecond_deviation=500000)

    replace_list (list_object, contents)

    setUp ()

    prepare_config ()

    create_file (content=None, dir=None, suffix="")
        Creates a test file on disk using tempfile.mkstemp() and uses the lower level file interfaces to man-
        age it. This is done to ensure we have more control of the file descriptor itself so on platforms such as
        Windows we don't have to worry about running out of file handles.

    create_directory (count=10)

class pyfarm.agent.testutil.BaseRequestTestCase (methodName='runTest')
    Bases: pyfarm.agent.testutil.TestCase

    HTTP_SCHEME = 'http'

    TEST_URL = 'http://httpbin.org'

    REDIRECT_TARGET = 'http://example.com'

    HTTP_REQUEST_SUCCESS = None

    setUp ()

class pyfarm.agent.testutil.BaseHTTPTestCase (methodName='runTest')
    Bases: pyfarm.agent.testutil.TestCase

```

```
URI = NotImplemented
CLASS = NotImplemented
CLASS_FACTORY = NotImplemented
DEFAULT_HEADERS = NotImplemented

setUp()

request(method, **kwargs)

instance_class()

test_instance()

test_leaf()

test_implements_methods()

test_methods_exist_for_schema()

class pyfarm.agent.testutil.BaseAPITestCase(methodName='runTest')
    Bases: pyfarm.agent.testutil.BaseHTTPTestCase

    DEFAULT_HEADERS = {'Accept': ['application/json']}

    test_parent()

class pyfarm.agent.testutil.BaseHTMLTestCase(methodName='runTest')
    Bases: pyfarm.agent.testutil.BaseHTTPTestCase

    DEFAULT_HEADERS = {'Accept': ['text/html']}

    test_template_set()

    test_template_loaded()
```

4.2.5 pyfarm.agent.utility module

Utilities

Top level utilities for the agent to use internally. Many of these are copied over from the master (which we can't import here).

`pyfarm.agent.utility.validate_environment(values)`

Ensures that values is a dictionary and that it only contains string keys and values.

`pyfarm.agent.utility.validate_uuid(value)`

Ensures that value can be converted to or is a UUID object.

`pyfarm.agent.utility.TASKS_SCHEMA(values)`

`pyfarm.agent.utility.default_json_encoder(obj, return_obj=False)`

`pyfarm.agent.utility.json_safe(source)`

Recursively converts source into something that should be safe for `json.dumps()` to handle. This is used in conjunction with `default_json_encoder()` to also convert keys to something the json encoder can understand.

`pyfarm.agent.utility.quote_url(source_url)`

This function serves as a wrapper around `urlsplit()` and `quote()` and a url that has the path quoted.

`pyfarm.agent.utility.dumps(*args, **kwargs)`

Agent's implementation of `json.dumps()` or `pyfarm.master.utility.jsonify()`

`pyfarm.agent.utility.request_from_master(request)`

Returns True if the request appears to be coming from the master

class `pyfarm.agent.utility.UTF8Recoder(f, encoding)`

Bases: `object`

Iterator that reads an encoded stream and reencodes the input to UTF-8

`next()`

class `pyfarm.agent.utility.UnicodeCSVReader(f, dialect=<class csv.excel>, encoding='utf-8', **kwargs)`

Bases: `object`

A CSV reader which will iterate over lines in the CSV file “f”, which is encoded in the given encoding.

`next()`

class `pyfarm.agent.utility.UnicodeCSVWriter(f, dialect=<class csv.excel>, **kwargs)`

Bases: `object`

A CSV writer which will write rows to CSV file “f”, which is encoded in the given encoding.

`writerow(row)`

`writerows(rows)`

`pyfarm.agent.utility.total_seconds(td)`

Returns the total number of seconds in the time delta object. This function is provided for backwards comparability with Python 2.6.

class `pyfarm.agent.utility.AgentUUID`

Bases: `object`

This class wraps all the functionality required to load, cache and retrieve an Agent’s UUID.

`log = <pyfarm.agent.logger.python.Logger object>`

classmethod `load(path)`

A classmethod to load a UUID object from a path. If the provided `path` does not exist or does not contain data which can be converted into a UUID object `None` will be returned.

classmethod `save(agent_uuid, path)`

Saves `agent_uuid` to `path`. This classmethod will also create the necessary parent directories and handle conversion from the input type `uuid.UUID`.

classmethod `generate()`

Generates a UUID object. This simply wraps `uuid.uuid4()` and logs a warning.

`pyfarm.agent.utility.remove_file(path, retry_on_exit=False, raise_=True, ignored_errnos=(2,))`

Simple function to remove the provided file or retry on exit if requested. This function standardizes the log output, ensures it’s only called once per path on exit and handles platform specific exceptions (ie. `WindowsError`).

Parameters

- **retry_on_exit** (*bool*) – If True, retry removal of the file when Python exists.
- **raise** (*bool*) – If True, raise an exceptions produced. This will always be False if `remove_file()` is being executed by `atexit`
- **ignored_errnos** (*tuple*) – A tuple of ignored error numbers. By default this function only ignores `ENOENT`.

```
pyfarm.agent.utility.remove_directory(path, retry_on_exit=False, raise_=True, ignored_errnos=(2, ))
```

Simple function to **recursively** remove the provided directory or retry on exit if requested. This function standardizes the log output, ensures it's only called once per path on exit and handles platform specific exceptions (ie. `WindowsError`).

Parameters

- **retry_on_exit** (*bool*) – If True, retry removal of the file when Python exists.
- **raise** (*bool*) – If True, raise an exceptions produced. This will always be False if `remove_directory()` is being executed by `atexit`
- **ignored_errnos** (*tuple*) – A tuple of ignored error numbers. By default this function only ignores `ENOENT`.

exception `pyfarm.agent.utility.LockTimeoutError`

Bases: `exceptions.Exception`

Raised if we timeout while attempting to acquire a deferred lock

class `pyfarm.agent.utility.TimedDeferredLock`

Bases: `twisted.internet.defer.DeferredLock`

A subclass of `DeferredLock` which has a timeout for the `acquire()` call.

acquire (*timeout=None*)

This method operates the same as `DeferredLock.acquire()` does except it requires a timeout argument.

Parameters **timeout** (*int*) – The number of seconds to wait before timing out.

Raises `LockTimeoutError` – Raised if the timeout was reached before we could acquire the lock.

4.3 Module contents

4.3.1 PyFarm Agent

Core module containing code to run PyFarm's agent.

5.1 Subpackages

5.1.1 pyfarm.jobtypes.core package

Submodules

pyfarm.jobtypes.core.internals module

Job Type Internals

Contains classes which contain internal methods for the `pyfarm.jobtypes.core.jobtype.JobType` class.

```
class pyfarm.jobtypes.core.internals.ProcessData (protocol, started, stopped)  
    Bases: tuple
```

```
    protocol  
        Alias for field number 0
```

```
    started  
        Alias for field number 1
```

```
    stopped  
        Alias for field number 2
```

```
exception pyfarm.jobtypes.core.internals.InsufficientSpaceError  
    Bases: exceptions.Exception
```

```
class pyfarm.jobtypes.core.internals.Cache  
    Bases: object
```

```
    Internal methods for caching job types
```

```
    cache = {}
```

```
    JOBTTYPE_VERSION_URL = '%(master_api)s/jobtypes/%(name)s/versions/%(version)s'
```

```
CACHE_DIRECTORY = '/tmp/pyfarm/agent/jobtype_cache'

e = OSError(17, 'File exists')

class pyfarm.jobtypes.core.internals.Process
    Bases: object

    Methods related to process control and management

    logging = {}

class pyfarm.jobtypes.core.internals.System
    Bases: object

    uuid = NotImplemented

class pyfarm.jobtypes.core.internals.TypeChecks
    Bases: object

    Helper static methods for performing type checks on input arguments.
```

pyfarm.jobtypes.core.jobtype module

Job Type Core

This module contains the core job type from which all other job types are built. All other job types must inherit from the *JobType* class in this module.

```
exception pyfarm.jobtypes.core.jobtype.TaskNotFound
    Bases: exceptions.Exception

exception pyfarm.jobtypes.core.jobtype.ConnectionBroken
    Bases: exceptions.Exception

class pyfarm.jobtypes.core.jobtype.CommandData (command, *arguments, **kwargs)
    Bases: object

    Stores data to be returned by JobType.get_command_data(). Instances of this class are also used by
    JobType.spawn_process_inputs() at execution time.
```

Note: This class does not perform any key of path resolution by default. It is assumed this has already been done using something like *JobType.map_path()*

Parameters

- **command** (*string*) – The command that will be executed when the process runs.
- **arguments** – Any additional arguments to be passed along to the command being launched.
- **env** (*dict*) – If provided, this will be the environment to launch the command with. If this value is not provided then a default environment will be setup using *set_default_environment()* when *JobType.start()* is called. *JobType.start()* itself will use *JobType.set_default_environment()* to generate the default environment.
- **cwd** (*string*) – The working directory the process should execute in. If not provided the process will execute in whatever the directory the agent is running inside of.

- **user** (*string* or *integer*) – The username or user id that the process should run as. On Windows this keyword is ignored and on Linux this requires the agent to be executing as root. The value provided here will be run through `JobType.get_uid_gid()` to map the incoming value to an integer.
- **group** (*string* or *integer*) – Same as `user` above except this sets the group the process will execute.
- **id** – An arbitrary id to associate with the resulting process protocol. This can help identify

validate()

Validates that the attributes on an instance of this class contain values we expect. This method is called externally by the job type in `JobType.start()` and may correct some instance attributes.

set_default_environment (*value*)

Sets the environment to *value* if the internal `env` attribute is `None`. By default this method is called by the job type and passed in the results from `pyfarm.jobtype.core.JobType.get_environment()`

class `pyfarm.jobtypes.core.jobtype.JobType` (*assignment*)

Bases: `pyfarm.jobtypes.core.internals.Cache`, `pyfarm.jobtypes.core.internals.System`, `pyfarm.jobtypes.core.internals.Process`, `pyfarm.jobtypes.core.internals.TypeChecks`

Base class for all other job types. This class is intended to abstract away many of the asynchronous necessary to run a job type on an agent.

Variables

- **PERSISTENT_JOB_DATA** (*set*) – A dictionary of job ids and data that `prepare_for_job()` has produced. This is used during `__init__()` to set `persistent_job_data`.
- **COMMAND_DATA_CLASS** (`CommandData`) – If you need to provide your own class to represent command data you should override this attribute. This attribute is used by methods within this class to do type checking.
- **PROCESS_PROTOCOL** (`ProcessProtocol`) – The protocol object used to communicate with each process spawned
- **ASSIGNMENT_SCHEMA** (`voluptuous.Schema`) – The schema of an assignment. This object helps to validate the incoming assignment to ensure it's not missing any data.
- **uuid** (*UUID*) – This is the unique identifier for the job type instance and is automatically set when the class is instantiated. This is used by the agent to track assignments and job type instances.
- **finished_tasks** (*set*) – A set of tasks that have had their state changed to finished through `set_task_state()`. At the start of the assignment, this list is empty.
- **failed_tasks** (*set*) – This is analogous to `finished_tasks` except it contains failed tasks only.

Parameters `assignment` (*dict*) – This attribute is a dictionary the keys “job”, “jobtype” and “tasks”. `self.assignment[“job”]` is itself a dict with keys “id”, “title”, “data”, “environ” and “by”. The most important of those is usually “data”, which is the dict specified when submitting the job and contains jobtype specific data. `self.assignment[“tasks”]` is a list of dicts representing the tasks in the current assignment. Each of these dicts has the keys “id” and “frame”. The list is ordered by frame number.

PERSISTENT_JOB_DATA = {}

COMMAND_DATA

alias of *CommandData*

PROCESS_PROTOCOL

alias of *ProcessProtocol*

ASSIGNMENT_SCHEMA = `<Schema({'job': <Schema({'cpus': Any([<type 'int'>, <type 'long'>])})>`

classmethod load (*assignment*)

Given an assignment this class method will load the job type either from cache or from the master.

Parameters *assignment* (*dict*) – The dictionary containing the assignment. This will be passed into an instance of **ASSIGNMENT_SCHEMA** to validate that the internal data is correct.

classmethod prepare_for_job (*job*)

Note: This method is not yet implemented

Called before a job executes on the agent first the first time. Whatever this classmethod returns will be available as *persistent_job_data* on the job type instance.

Parameters *job* (*int*) – The job id which *prepare_for_job* is being run for

By default this method does nothing.

classmethod cleanup_after_job (*persistent_data*)

Note: This method is not yet implemented

This classmethod will be called after the last assignment from a given job has finished on this node.

Parameters *persistent_data* – The persistent data that *prepare_for_job()* produced. The value for this data may be None if *prepare_for_job()* returned None or was not implemented.

classmethod spawn_persistent_process (*job, command_data*)

Note: This method is not yet implemented

Starts one child process using an instance of *CommandData* or similiar input. This process is intended to keep running until the last task from this job has been processed, potentially spanning more than one assignment. If the spawned process is still running then we'll cleanup the process after *cleanup_after_job()*

node ()

Returns live information about this host, the operating system, hardware, and several other pieces of global data which is useful inside of the job type. Currently data from this method includes:

- **master_api** - The base url the agent is using to communicate with the master.
- **hostname** - The hostname as reported to the master.
- **agent_id** - The unique identifier used to identify. this agent to the master.

- **id** - The database id of the agent as given to us by the master on startup of the agent.
- **cpus** - The number of CPUs reported to the master
- **ram** - The amount of ram reported to the master.
- **total_ram** - The amount of ram, in megabytes, that's installed on the system regardless of what was reported to the master.
- **free_ram** - How much ram, in megabytes, is free for the entire system.
- **consumed_ram** - How much ram, in megabytes, is being consumed by the agent and any processes it has launched.
- **admin** - Set to True if the current user is an administrator or 'root'.
- **user** - The username of the current user.
- **case_sensitive_files** - True if the file system is case sensitive.
- **case_sensitive_env** - True if environment variables are case sensitive.
- **machine_architecture** - The architecture of the machine the agent is running on. This will return 32 or 64.
- **operating_system** - The operating system the agent is executing on. This value will be 'linux', 'mac' or 'windows'. In rare circumstances this could also be 'other'.

Raises **KeyError** – Raised if one or more keys are not present in the global configuration object.

This should rarely if ever be a problem under normal circumstances. The exception to this rule is in unittests or standalone libraries with the global config object may not be populated.

assignments()

Short cut method to access tasks

tempdir (*new=False, remove_on_finish=True*)

Returns a temporary directory to be used within a job type. By default once called the directory will be created on disk and returned from this method.

Calling this method multiple times will return the same directory instead of creating a new directory unless *new* is set to True.

Parameters

- **new** (*bool*) – If set to True then return a new directory when called. This however will not replace the 'default' temp directory.
- **remove_on_finish** (*bool*) – If True then keep track of the directory returned so it can be removed when the job type finishes.

get_uid_gid (*user, group*)

Overridable. This method to convert a named user and group into their respective user and group ids.

get_environment ()

Constructs an environment dictionary that can be used when a process is spawned by a job type.

get_command_list (*commands*)

Convert a list of commands to a tuple with any environment variables expanded.

Parameters **commands** (*list*) – A list of strings to expand. Each entry in list will be passed into and returned from *expandvars()*.

Raises **TypeError** – Raised if *commands* is not a list or tuple.

Return type `tuple`

Returns Returns the expanded list of commands.

get_csvlog_path (*protocol_uuid*, *create_time=None*)

Returns the path to the comma separated value (csv) log file. The agent stores logs from processes in a csv format so we can store additional information such as a timestamp, line number, stdout/stderr identification and the the log message itself.

Note: This method should not attempt to create the parent directories of the resulting path. This is already handled by the logger pool in a non-blocking fashion.

Parameters

- **protocol_uuid** (*uuid.UUID*) – The UUID of the job type’s protocol instance.
- **create_time** (*datetime.datetime*) – If provided then the create time of the log file will equal this value. Otherwise this method will use the current UTC time for `create_time`

Raises `TypeError` – Raised if `protocol_uuid` or `create_time` are not the correct types.

get_command_data ()

Overridable. This method returns the arguments necessary for executing a command. For job types which execute a single process per assignment, this is the most important method to implement.

Warning: This method should not be used when this jobtype requires more than one process for one assignment and may not get called at all if `start()` was overridden.

The default implementation does nothing. When overriding this method you should return an instance of `COMMAND_DATA_CLASS`:

```
return self.COMMAND_DATA(  
    "/usr/bin/python", "-c", "print 'hello world'",  
    env={"FOO": "bar"}, user="bob")
```

See `CommandData`’s class documentation for a full description of possible arguments.

Please note however the default command data class, `CommandData` does not perform path expansion. So instead you have to handle this yourself with `map_path()`.

map_path (*path*)

Takes a string argument. Translates a given path for any OS to what it should be on this particular node. This does not communicate with the master.

Parameters **path** (*string*) – The path to translate to an OS specific path for this node.

Raises `TypeError` – Raised if `path` is not a string.

expandvars (*value*, *environment=None*, *expand=None*)

Expands variables inside of a string using an environment.

Parameters

- **value** (*string*) – The path to expand.
- **environment** (*dict*) – The environment to use for expanding `value`. If this value is `None` (the default) then we’ll use `get_environment()` to build this value.

- **expand** (*bool*) – When not provided we use the `jobtype_expandvars` configuration value to set the default. When this value is `True` we'll perform environment variable expansion otherwise we return `value` untouched.

start ()

This method is called when the job type should start working. Depending on the job type's implementation this will prepare and start one more more processes.

stop (*assignment_failed=False, avoid_reassignment=False, error=None, signal='KILL'*)

This method is called when the job type should stop running. This will terminate any processes associated with this job type and also inform the master of any state changes to an associated task or tasks.

Parameters

- **assignment_failed** (*boolean*) – Whether this means the assignment has genuinely failed. By default, we assume that stopping this assignment was the result of deliberate user action (like stopping the job or shutting down the agent), and won't treat it as a failed assignment.
- **avoid_reassignment** (*boolean*) – If set to true, the agent will add itself to the lists of agents that failed the tasks in this assignment. Can be useful when we want to return the assignment to the master without increasing its failures counter, but still don't want it to be reassigned to us.
- **error** (*string*) – If the assignment has failed, this string is upload as `last_error` for the failed tasks.
- **signal** (*string*) – The signal to send the any running processes. Valid options are `KILL`, `TERM` or `INT`.

format_error (*error*)

Takes some kind of object, typically an instance of `Exception` or `:class'Failure'`, and produces a human readable string.

Return type *string* or *None*

Returns Returns a string if we know how to format the error. Otherwise this method returns `None` and logs an error.

set_states (*tasks, state, error=None*)

Wrapper around `set_state()` that that allows you to the state on the master for multiple tasks at once.

set_task_progress (**args, **kwargs*)

Sets the progress of the given task

Parameters

- **task** (*dict*) – The dictionary containing the task we're changing the progress for.
- **progress** (*float*) – The progress to set on `task`

add_self_to_failed_on_agents (**args, **kwargs*)

Adds this agent to the list of agents that failed to execute the given task, without explicitly setting this task to failed.

Parameters **task** (*dict*) – The dictionary containing the task

set_task_started_now (**args, **kwargs*)

Sets the `time_started` of the given task to the current time on the master.

This method is useful for batched tasks, where the actual work on a single task may start much later than the work the assignment as a whole.

Parameters `task` (*dict*) – The dictionary containing the task we’re changing the start time for.

set_task_state (**args, **kwargs*)

Sets the state of the given task

Parameters

- **task** (*dict*) – The dictionary containing the task we’re changing the state for.
- **state** (*string*) – The state to change task to
- **error** (*string, Exception*) – If the state is changing to ‘error’ then also set the `last_error` column. Any exception instance that is passed to this keyword will be passed through `format_exception()` first to format it.

get_local_task_state (*task_id*)

Returns None if the state of this task has not been changed locally since this assignment has started. This method does not communicate with the master.

is_successful (*protocol, reason*)

Overridable. This method that determines whether the process referred to by a protocol instance has exited successfully.

The default implementation returns True if the process’s return code was 0 and False` in all other cases. If you need to modify this behavior please be aware that ``reason may be an integer or an instance of `twisted.internet.error.ProcessTerminated` if the process terminated without errors or an instance of `twisted.python.failure.Failure` if there were problems.

Raises `NotImplementedError` – Raised if we encounter a condition that the base implementation is unable to handle.

before_start ()

Overridable. This method called directly before `start()` itself is called.

The default implementation does nothing and values returned from this method are ignored.

before_spawn_process (*command, protocol*)

Overridable. This method called directly before a process is spawned.

By default this method does nothing except log information about the command we’re about to launch both the the agent’s log and to the log file on disk.

Parameters

- **command** (*CommandData*) – An instance of `CommandData` which contains the environment to use, command and arguments. Modifications to this object will be applied to the process being spawned.
- **protocol** (*ProcessProtocol*) – An instance of `pyfarm.jobtypes.core.process.ProcessProtocol` which contains the protocol used to communicate between the process and this job type.

process_stopped (*protocol, reason*)

Overridable. This method called when a child process stopped running.

The default implementation will mark all tasks in the current assignment as done or failed of there was at least one failed process.

process_started (*protocol*)

Overridable. This method is called when a child process started running.

The default implementation will mark all tasks in the current assignment as running.

process_output (*protocol, output, line_fragments, line_handler*)

This is a mid-level method which takes output from a process protocol then splits and processes it to ensure we pass complete output lines to the other methods.

Implementors who wish to process the output line by line should override `preprocess_stdout_line()`, `preprocess_stdout_line()`, `process_stdout_line()` or `process_stderr_line()` instead. This method is a glue method between other parts of the job type and should only be overridden if there's a problem or you want to change how lines are split.

Parameters

- **protocol** (*ProcessProtocol*) – The protocol instance which produced output
- **output** (*string*) – The blob of text or line produced
- **line_fragments** (*dict*) – The line fragment dictionary containing individual line fragments. This will be either `self._stdout_line_fragments` or `self._stderr_line_fragments`.
- **line_handler** (*callable*) – The function to handle any lines produced. This will be either `handle_stdout_line()` or `handle_stderr_line()`

Returns This method returns nothing by default and any return value produced by this method will not be consumed by other methods.

handle_stdout_line (*protocol, stdout*)

Takes a *ProcessProtocol* instance and `stdout` line produced by `process_output()` and runs it through all the steps necessary to preprocess, format, log and handle the line.

The default implementation will run `stdout` through several methods in order:

- `preprocess_stdout_line()`
- `format_stdout_line()`
- `log_stdout_line()`
- `process_stdout_line()`

Warning: This method is not private however it's advisable to override the methods above instead of this one. Unlike this method, which is more generalized and invokes several other methods, the above provide more targeted functionality.

Parameters

- **protocol** (*ProcessProtocol*) – The protocol instance which produced `stdout`
- **stderr** (*string*) – A complete line to `stderr` being emitted by the process

Returns This method returns nothing by default and any return value produced by this method will not be consumed by other methods.

handle_stderr_line (*protocol, stderr*)

Overridable. Takes a *ProcessProtocol* instance and `stderr` produced by `process_output()` and runs it through all the steps necessary to preprocess, format, log and handle the line.

The default implementation will run `stderr` through several methods in order:

- `preprocess_stderr_line()`
- `format_stderr_line()`

- `log_stderr_line()`
- `process_stderr_line()`

Warning: This method is overridable however it's advisable to override the methods above instead. Unlike this method, which is more generalized and invokes several other methods, the above provide more targeted functionality.

Parameters

- **protocol** (*ProcessProtocol*) – The protocol instance which produced stdout
- **stderr** (*string*) – A complete line to stderr being emitted by the process

Returns This method returns nothing by default and any return value produced by this method will not be consumed by other methods.

preprocess_stdout_line (*protocol, stdout*)

Overridable. Provides the ability to manipulate stdout or protocol before it's passed into any other line handling methods.

The default implementation does nothing.

Parameters

- **protocol** (*ProcessProtocol*) – The protocol instance which produced stdout
- **stderr** (*string*) – A complete line to stdout before any formatting or logging has occurred.

Return type *string*

Returns This method returns nothing by default but when overridden should return a string which will be used in line handling methods such as `format_stdout_line()`, `log_stdout_line()` and `process_stdout_line()`.

preprocess_stderr_line (*protocol, stderr*)

Overridable. Formats a line from stdout before it's passed onto methods such as `log_stdout_line()` and `process_stdout_line()`.

The default implementation does nothing.

Parameters

- **protocol** (*ProcessProtocol*) – The protocol instance which produced stderr
- **stderr** (*string*) – A complete line to stderr before any formatting or logging has occurred.

Return type *string*

Returns This method returns nothing by default but when overridden should return a string which will be used in line handling methods such as `format_stderr_line()`, `log_stderr_line()` and `process_stderr_line()`.

format_stdout_line (*protocol, stdout*)

Overridable. Formats a line from stdout before it's passed onto methods such as `log_stdout_line()` and `process_stdout_line()`.

The default implementation does nothing.

Parameters

- **protocol** (*ProcessProtocol*) – The protocol instance which produced stdout
- **stdout** (*string*) – A complete line from process to format and return.

Return type *string*

Returns This method returns nothing by default but when overridden should return a string which will be used in *log_stdout_line()* and *process_stdout_line()*

format_stderr_line (*protocol, stderr*)

Overridable. Formats a line from stderr before it's passed onto methods such as *log_stderr_line()* and *process_stderr_line()*.

The default implementation does nothing.

Parameters

- **protocol** (*ProcessProtocol*) – The protocol instance which produced stderr
- **stderr** (*string*) – A complete line from the process to format and return.

Return type *string*

Returns This method returns nothing by default but when overridden should return a string which will be used in *log_stderr_line()* and *process_stderr_line()*

log_stdout_line (*protocol, stdout*)

Overridable. Called when we receive a complete line on stdout from the process.

The default implementation will use the global logging pool to log stdout to a file.

Parameters

- **protocol** (*ProcessProtocol*) – The protocol instance which produced stdout
- **stderr** (*string*) – A complete line to stdout that has been formatted and is ready to log to a file.

Returns This method returns nothing by default and any return value produced by this method will not be consumed by other methods.

log_stderr_line (*protocol, stderr*)

Overridable. Called when we receive a complete line on stderr from the process.

The default implementation will use the global logging pool to log stderr to a file.

Parameters

- **protocol** (*ProcessProtocol*) – The protocol instance which produced stderr
- **stderr** (*string*) – A complete line to stderr that has been formatted and is ready to log to a file.

Returns This method returns nothing by default and any return value produced by this method will not be consumed by other methods.

process_stderr_line (*protocol, stderr*)

Overridable. This method is called when we receive a complete line to stderr. The line will be preformatted and will already have been sent for logging.

The default implementation sends “stderr” and “protocol” to :meth:‘process_stdout_line‘.

Parameters

- **protocol** (*ProcessProtocol*) – The protocol instance which produced stderr
- **stderr** (*string*) – A complete line to stderr after it has been formatted and logged.

Returns This method returns nothing by default and any return value produced by this method will not be consumed by other methods.

process_stdout_line (*protocol*, *stdout*)

Overridable. This method is called when we receive a complete line to `stdout`. The line will be preformatted and will already have been sent for logging.

The default implementation does nothing.

Parameters

- **protocol** (*ProcessProtocol*) – The protocol instance which produced `stderr`
- **stdout** (*string*) – A complete line to `stdout` after it has been formatted and logged.

Returns This method returns nothing by default and any return value produced by this method will not be consumed by other methods.

pyfarm.jobtypes.core.process module

Process

Module responsible for connecting a Twisted process object and a job type. Additionally this module contains other classes which are useful in starting or managing a process.

class `pyfarm.jobtypes.core.process.ReplaceEnvironment` (*frozen_environment*, *environment=None*)

Bases: `object`

A context manager which will replace `os.environ`'s, or dictionary of your choosing, for a short period of time. After exiting the context manager the original environment will be restored.

This is useful if you have something like a process that's using global environment and you want to ensure that global environment is always consistent.

Parameters **environment** (*dict*) – If provided, use this as the environment dictionary instead of `os.environ`

class `pyfarm.jobtypes.core.process.ProcessProtocol` (*jobtype*)

Bases: `twisted.internet.protocol.ProcessProtocol`

Subclass of `Protocol` which hooks into the various systems necessary to run and manage a process. More specifically, this helps to act as plumbing between the process being run and the job type.

uuid

pid

process

The underlying Twisted process object

psutil_process

Returns a `psutil.Process` object for the running process

connectionMade ()

Called when the process first starts and the file descriptors have opened.

processEnded (*reason*)

Called when the process has terminated and all file descriptors have been closed. `processExited()` is called, too, however we only want to notify the parent job type once the process has freed up the last bit of resources.

outReceived (*data*)

Called when the process emits on stdout

errReceived (*data*)

Called when the process emits on stderr

kill ()

Kills the underlying process, if running.

terminate ()

Terminates the underlying process, if running.

interrupt ()

Interrupts the underlying process, if running.

running ()

Method to determine whether the child process is currently running

Module contents

5.2 Module contents

5.2.1 Job Types

This package, *pyfarm.jobtypes*, contains the code which executes a task on an agent.

CHAPTER 6

Indices and tables

- `genindex`
- `modindex`
- `search`

HTTP Routing Table

/api

GET /api/v1/tasklogs/<identifier>
HTTP/1.1, 24
GET /api/v1/tasks/ HTTP/1.1, 25
GET /api/v1/versions/ HTTP/1.1, 22
POST /api/v1/check_software HTTP/1.1,
23
POST /api/v1/update HTTP/1.1, 26
DELETE /api/v1/tasks/<int:task_id>
HTTP/1.1, 25

p

- `pyfarm.agent`, 44
- `pyfarm.agent.config`, 36
- `pyfarm.agent.entrypoints`, 21
- `pyfarm.agent.entrypoints.development`, 19
- `pyfarm.agent.entrypoints.main`, 19
- `pyfarm.agent.entrypoints.parser`, 20
- `pyfarm.agent.entrypoints.supervisor`, 21
- `pyfarm.agent.entrypoints.utility`, 21
- `pyfarm.agent.http`, 32
 - `pyfarm.agent.http.api`, 26
 - `pyfarm.agent.http.api.assign`, 22
 - `pyfarm.agent.http.api.base`, 22
 - `pyfarm.agent.http.api.config`, 23
 - `pyfarm.agent.http.api.software`, 23
 - `pyfarm.agent.http.api.state`, 24
 - `pyfarm.agent.http.api.tasklogs`, 24
 - `pyfarm.agent.http.api.tasks`, 25
 - `pyfarm.agent.http.api.update`, 26
 - `pyfarm.agent.http.core`, 31
 - `pyfarm.agent.http.core.client`, 26
 - `pyfarm.agent.http.core.resource`, 29
 - `pyfarm.agent.http.core.server`, 30
 - `pyfarm.agent.http.core.template`, 31
 - `pyfarm.agent.http.system`, 31
 - `pyfarm.agent.manhole`, 37
 - `pyfarm.agent.service`, 38
- `pyfarm.agent.sysinfo`, 36
 - `pyfarm.agent.sysinfo.cpu`, 32
 - `pyfarm.agent.sysinfo.disks`, 33
 - `pyfarm.agent.sysinfo.graphics`, 33
 - `pyfarm.agent.sysinfo.memory`, 33
 - `pyfarm.agent.sysinfo.network`, 34
 - `pyfarm.agent.sysinfo.software`, 34
 - `pyfarm.agent.sysinfo.system`, 35
 - `pyfarm.agent.sysinfo.user`, 35
- `pyfarm.agent.testutil`, 40
- `pyfarm.agent.utility`, 42
- `pyfarm.jobtypes`, 57
 - `pyfarm.jobtypes.core`, 57
 - `pyfarm.jobtypes.core.internals`, 45
 - `pyfarm.jobtypes.core.jobtype`, 46
 - `pyfarm.jobtypes.core.process`, 56

Symbols

`_expandvars()` (pyfarm.agent.config.LoggingConfiguration method), 36

A

`acquire()` (pyfarm.agent.utility.TimedDeferredLock method), 44

`ActionMixin` (class in pyfarm.agent.entrypoints.parser), 20

`add_self_to_failed_on_agents()` (pyfarm.jobtypes.core.jobtype.JobType method), 51

`addresses()` (in module pyfarm.agent.sysinfo.network), 34

`Agent` (class in pyfarm.agent.service), 38

`agent_api` (pyfarm.agent.entrypoints.main.AgentEntryPoint attribute), 19

`agent_api()` (pyfarm.agent.service.Agent class method), 38

`AgentArgumentParser` (class in pyfarm.agent.entrypoints.parser), 21

`AgentEntryPoint` (class in pyfarm.agent.entrypoints.main), 19

`agents_endpoint()` (pyfarm.agent.service.Agent class method), 38

`AgentUUID` (class in pyfarm.agent.utility), 43

`ALLOW_DIRECTORY_LISTING` (pyfarm.agent.http.core.server.StaticPath attribute), 30

`ALLOWED_ACCEPT` (pyfarm.agent.http.api.base.APIResource attribute), 22

`ALLOWED_ACCEPT` (pyfarm.agent.http.core.resource.Resource attribute), 29

`ALLOWED_ACCEPT` (pyfarm.agent.http.system.HTMLResource attribute), 31

`ALLOWED_CONTENT_TYPE` (pyfarm.agent.http.api.base.APIResource attribute), 22

(pyfarm.agent.http.core.resource.Resource attribute), 30

`ALLOWED_CONTENT_TYPE` (pyfarm.agent.http.system.HTMLResource attribute), 31

`APIResource` (class in pyfarm.agent.http.api.base), 22

`APIRoot` (class in pyfarm.agent.http.api.base), 22

`APITestServer` (class in pyfarm.agent.testutil), 40

`APITestServerResource` (class in pyfarm.agent.testutil), 40

`AppendAction` (in module pyfarm.agent.entrypoints.parser), 21

`AppendConstAction` (in module pyfarm.agent.entrypoints.parser), 21

`assert_parser()` (in module pyfarm.agent.entrypoints.parser), 20

`assertDateAlmostEqual()` (pyfarm.agent.testutil.TestCase method), 41

`assertRaises()` (pyfarm.agent.testutil.TestCase method), 41

`assertRaisesRegexp()` (pyfarm.agent.testutil.TestCase method), 41

`Assign` (class in pyfarm.agent.http.api.assign), 22

`ASSIGNMENT_SCHEMA` (pyfarm.jobtypes.core.jobtype.JobType attribute), 48

`assignments()` (pyfarm.jobtypes.core.jobtype.JobType method), 49

B

`BaseAPITestCase` (class in pyfarm.agent.testutil), 42

`BaseHTMLTestCase` (class in pyfarm.agent.testutil), 42

`BaseHTTPTestCase` (class in pyfarm.agent.testutil), 41

`BaseRequestTestCase` (class in pyfarm.agent.testutil), 41

`before_spawn_process()` (pyfarm.jobtypes.core.jobtype.JobType method), 52

`before_start()` (pyfarm.jobtypes.core.jobtype.JobType method), 52
`build_http_resource()` (pyfarm.agent.service.Agent method), 39
`build_url()` (in module pyfarm.agent.http.core.client), 27

C

`Cache` (class in pyfarm.jobtypes.core.internals), 45
`cache` (pyfarm.agent.http.core.template.InMemoryCache attribute), 31
`cache` (pyfarm.jobtypes.core.internals.Cache attribute), 45
`CACHE_DIRECTORY` (pyfarm.jobtypes.core.internals.Cache attribute), 46
`callback_agent_id_set()` (pyfarm.agent.service.Agent method), 39
`callback_shutting_down_changed()` (pyfarm.agent.service.Agent method), 40
`callbacks` (pyfarm.agent.config.ConfigurationWithCallbacks attribute), 37
`changed()` (pyfarm.agent.config.ConfigurationWithCallbacks method), 37
`changed()` (pyfarm.agent.config.LoggingConfiguration method), 37
`check_software_availability()` (in module pyfarm.agent.sysinfo.software), 35
`CheckSoftware` (class in pyfarm.agent.http.api.software), 23
`CLASS` (pyfarm.agent.testutil.BaseHTTPTestCase attribute), 42
`CLASS_FACTORY` (pyfarm.agent.testutil.BaseHTTPTestCase attribute), 42
`cleanup_after_job()` (pyfarm.jobtypes.core.jobtype.JobType class method), 48
`clear()` (pyfarm.agent.config.ConfigurationWithCallbacks method), 37
`clear()` (pyfarm.agent.config.LoggingConfiguration method), 37
`clear()` (pyfarm.agent.http.core.template.InMemoryCache method), 31
`code` (pyfarm.agent.testutil.DummyRequest attribute), 40
`COMMAND_DATA` (pyfarm.jobtypes.core.jobtype.JobType attribute), 47
`CommandData` (class in pyfarm.jobtypes.core.jobtype), 46
`Config` (class in pyfarm.agent.http.api.config), 23
`Configuration` (class in pyfarm.agent.http.system), 32
`ConfigurationWithCallbacks` (class in pyfarm.agent.config), 37

`configure_logger_level()` (in module pyfarm.agent.config), 37
`ConnectionBroken`, 46
`connectionLost()` (pyfarm.agent.http.core.client.Response method), 28
`connectionLost()` (pyfarm.agent.manhole.LoggingManhole method), 38
`connectionMade()` (pyfarm.agent.manhole.LoggingManhole method), 38
`connectionMade()` (pyfarm.jobtypes.core.process.ProcessProtocol method), 56
`cpu_name()` (in module pyfarm.agent.sysinfo.cpu), 32
`create_directory()` (pyfarm.agent.testutil.TestCase method), 41
`create_file()` (pyfarm.agent.testutil.TestCase method), 41
`create_jobtype()` (in module pyfarm.agent.testutil), 40
`CREATED` (pyfarm.agent.config.LoggingConfiguration attribute), 36

D

`data()` (pyfarm.agent.http.core.client.Response method), 28
`dataReceived()` (pyfarm.agent.http.core.client.Response method), 28
`DEFAULT_ACCEPT` (pyfarm.agent.http.api.base.APIResource attribute), 22
`DEFAULT_ACCEPT` (pyfarm.agent.http.core.resource.Resource attribute), 30
`DEFAULT_CONTENT_TYPE` (pyfarm.agent.http.api.base.APIResource attribute), 22
`DEFAULT_CONTENT_TYPE` (pyfarm.agent.http.core.resource.Resource attribute), 30
`DEFAULT_HEADERS` (pyfarm.agent.testutil.BaseAPITestCase attribute), 42
`DEFAULT_HEADERS` (pyfarm.agent.testutil.BaseHTMLTestCase attribute), 42
`DEFAULT_HEADERS` (pyfarm.agent.testutil.BaseHTTPTestCase attribute), 42
`default_json_encoder()` (in module pyfarm.agent.utility), 42
`delete()` (pyfarm.agent.http.api.tasks.Tasks method), 25
`DELETED` (pyfarm.agent.config.LoggingConfiguration attribute), 37
`deregister_callback()` (pyfarm.agent.config.ConfigurationWithCallbacks class method), 37

- directoryListing() (pyfarm.agent.http.core.server.StaticPath method), 31
- direxists() (in module pyfarm.agent.entrypoints.parser), 20
- DiskInfo (class in pyfarm.agent.sysinfo.disks), 33
- disks() (in module pyfarm.agent.sysinfo.disks), 33
- displayTracebacks (pyfarm.agent.http.core.server.Site attribute), 30
- DummyRequest (class in pyfarm.agent.testutil), 40
- dump_bytecode() (pyfarm.agent.http.core.template.InMemoryCache method), 31
- dumps() (in module pyfarm.agent.utility), 42
- ## E
- e (pyfarm.jobtypes.core.internals.Cache attribute), 46
- EDITABLE_FIELDS (pyfarm.agent.http.system.Configuration attribute), 32
- enum() (in module pyfarm.agent.entrypoints.parser), 20
- Environment (class in pyfarm.agent.http.core.template), 31
- environment variable
- PYFARM_JOBTYPE_ALLOW_CODE_EXECUTION_IN_MODULE_ROOT, 11
 - PYFARM_JOBTYPE_SUBCLASSES_BASE_CLASS, 11
- environment_is_case_sensitive() (in module pyfarm.agent.sysinfo.system), 35
- error() (pyfarm.agent.http.core.client.HTTPLog static method), 27
- error() (pyfarm.agent.http.core.resource.Resource method), 30
- error() (pyfarm.agent.testutil.ErrorCapturingParser method), 40
- ErrorCapturingParser (class in pyfarm.agent.testutil), 40
- errReceived() (pyfarm.jobtypes.core.process.ProcessProtocol method), 57
- expandvars() (pyfarm.jobtypes.core.jobtype.JobType method), 50
- EXPIRES (pyfarm.agent.http.core.server.StaticPath attribute), 30
- ## F
- fake_render() (in module pyfarm.agent.entrypoints.development), 19
- fake_work() (in module pyfarm.agent.entrypoints.development), 19
- FakeAgent (class in pyfarm.agent.testutil), 40
- FakeRequestHeaders (class in pyfarm.agent.testutil), 40
- fileexists() (in module pyfarm.agent.entrypoints.parser), 20
- filesystem_is_case_sensitive() (in module pyfarm.agent.sysinfo.system), 35
- format_error() (pyfarm.jobtypes.core.jobtype.JobType method), 51
- format_stderr_line() (pyfarm.jobtypes.core.jobtype.JobType method), 55
- format_stdout_line() (pyfarm.jobtypes.core.jobtype.JobType method), 54
- free (pyfarm.agent.sysinfo.disks.DiskInfo attribute), 33
- freeCache() (in module pyfarm.agent.sysinfo.memory), 33
- ## G
- generate() (pyfarm.agent.utility.AgentUUID class method), 43
- get() (pyfarm.agent.http.api.base.Versions method), 23
- get() (pyfarm.agent.http.api.config.Config method), 23
- get() (pyfarm.agent.http.api.state.Status method), 24
- get() (pyfarm.agent.http.api.tasklogs.TaskLogs method), 24
- get() (pyfarm.agent.http.api.tasks.Tasks method), 25
- get() (pyfarm.agent.http.system.Configuration method), 32
- get() (pyfarm.agent.http.system.Index method), 31
- get_accept() (pyfarm.agent.http.core.resource.Resource method), 30
- get_command_data() (pyfarm.jobtypes.core.jobtype.JobType method), 50
- get_command_list() (pyfarm.jobtypes.core.jobtype.JobType method), 49
- get_content_type() (pyfarm.agent.http.core.resource.Resource method), 30
- get_csvlog_path() (pyfarm.jobtypes.core.jobtype.JobType method), 50
- get_discovery_code() (in module pyfarm.agent.sysinfo.software), 34
- get_environment() (pyfarm.jobtypes.core.jobtype.JobType method), 49
- get_local_task_state() (pyfarm.jobtypes.core.jobtype.JobType method), 52
- get_software_version_data() (in module pyfarm.agent.sysinfo.software), 34
- get_uid_gid() (pyfarm.jobtypes.core.jobtype.JobType method), 49
- getHeader() (pyfarm.agent.testutil.DummyRequest method), 41
- getRawHeaders() (pyfarm.agent.testutil.FakeRequestHeaders method), 40
- GPULookupError, 33

graphics_cards() (in module pyfarm.agent.sysinfo.graphics), 33

H

handle() (pyfarm.agent.testutil.APITestServerResource method), 40

handle() (pyfarm.agent.testutil.TestCaseLogHandler method), 41

handle_stderr_line() (pyfarm.jobtypes.core.jobtype.JobType method), 53

handle_stdout_line() (pyfarm.jobtypes.core.jobtype.JobType method), 53

HIDDEN_FIELDS (pyfarm.agent.http.system.Configuration attribute), 32

hostname() (in module pyfarm.agent.sysinfo.network), 34

HTMLResource (class in pyfarm.agent.http.system), 31

HTTP_REQUEST_SUCCESS (pyfarm.agent.testutil.BaseRequestTestCase attribute), 41

http_retry_delay() (in module pyfarm.agent.http.core.client), 27

HTTP_SCHEME (pyfarm.agent.testutil.BaseRequestTestCase attribute), 41

HTTPLog (class in pyfarm.agent.http.core.client), 26

I

idle_time() (in module pyfarm.agent.sysinfo.cpu), 32

Index (class in pyfarm.agent.http.system), 31

InMemoryCache (class in pyfarm.agent.http.core.template), 31

instance_class() (pyfarm.agent.testutil.BaseHTTPTestCase method), 42

InsufficientSpaceError, 45

interfaces() (in module pyfarm.agent.sysinfo.network), 34

interpreter_architecture() (in module pyfarm.agent.sysinfo.system), 35

interrupt() (pyfarm.jobtypes.core.process.ProcessProtocol method), 57

iowait() (in module pyfarm.agent.sysinfo.cpu), 32

ip() (in module pyfarm.agent.entrypoints.parser), 20

is_administrator() (in module pyfarm.agent.sysinfo.user), 35

is_successful() (pyfarm.jobtypes.core.jobtype.JobType method), 52

isLeaf (pyfarm.agent.http.api.assign.Assign attribute), 22

isLeaf (pyfarm.agent.http.api.base.APIResource attribute), 22

isLeaf (pyfarm.agent.http.api.base.APIRoot attribute), 22

isLeaf (pyfarm.agent.http.api.base.Versions attribute), 23

isLeaf (pyfarm.agent.http.api.config.Config attribute), 23

isLeaf (pyfarm.agent.http.api.software.CheckSoftware attribute), 23

isLeaf (pyfarm.agent.http.api.state.Restart attribute), 24

isLeaf (pyfarm.agent.http.api.state.Status attribute), 24

isLeaf (pyfarm.agent.http.api.state.Stop attribute), 24

isLeaf (pyfarm.agent.http.api.update.Update attribute), 26

isLeaf (pyfarm.agent.testutil.APITestServerResource attribute), 40

J

JobType (class in pyfarm.jobtypes.core.jobtype), 47

JOBTYPES_VERSION_URL (pyfarm.jobtypes.core.internals.Cache attribute), 45

json() (pyfarm.agent.http.core.client.Response method), 28

json_safe() (in module pyfarm.agent.utility), 42

K

kill() (pyfarm.jobtypes.core.process.ProcessProtocol method), 57

L

lineReceived() (pyfarm.agent.manhole.LoggingManhole method), 38

load() (in module pyfarm.agent.sysinfo.cpu), 32

load() (pyfarm.agent.utility.AgentUUID class method), 43

load() (pyfarm.jobtypes.core.jobtype.JobType class method), 48

load_bytecode() (pyfarm.agent.http.core.template.InMemoryCache method), 31

LockTimeoutError, 44

log (pyfarm.agent.utility.AgentUUID attribute), 43

log_stderr_line() (pyfarm.jobtypes.core.jobtype.JobType method), 55

log_stdout_line() (pyfarm.jobtypes.core.jobtype.JobType method), 55

logging (pyfarm.jobtypes.core.internals.Process attribute), 46

LoggingConfiguration (class in pyfarm.agent.config), 36

LoggingManhole (class in pyfarm.agent.manhole), 37

longMessage (pyfarm.agent.testutil.TestCase attribute), 41

M

mac_addresses() (in module pyfarm.agent.sysinfo.network), 34

machine_architecture() (in module pyfarm.agent.sysinfo.system), 35

manhole_factory() (in module pyfarm.agent.manhole), 38

map_path() (pyfarm.jobtypes.core.jobtype.JobType method), 50

- [master_contacted\(\)](#) (pyfarm.agent.config.LoggingConfiguration method), [37](#)
[maxDiff](#) (pyfarm.agent.testutil.TestCase attribute), [41](#)
[mb\(\)](#) (in module pyfarm.agent.http.system), [31](#)
[methods\(\)](#) (pyfarm.agent.http.core.resource.Resource method), [30](#)
[mix_action\(\)](#) (in module pyfarm.agent.entrypoints.parser), [20](#)
[MODIFIED](#) (pyfarm.agent.config.LoggingConfiguration attribute), [36](#)
[mountpoint](#) (pyfarm.agent.sysinfo.disks.DiskInfo attribute), [33](#)
- ## N
- [NAMESPACE](#) (pyfarm.agent.manhole.TelnetRealm attribute), [38](#)
[next\(\)](#) (pyfarm.agent.utility.UnicodeCSVReader method), [43](#)
[next\(\)](#) (pyfarm.agent.utility.UTF8Recoder method), [43](#)
[node\(\)](#) (pyfarm.jobtypes.core.jobtype.JobType method), [48](#)
[number\(\)](#) (in module pyfarm.agent.entrypoints.parser), [20](#)
- ## O
- [operating_system\(\)](#) (in module pyfarm.agent.sysinfo.system), [35](#)
[outReceived\(\)](#) (pyfarm.jobtypes.core.process.ProcessProtocol method), [56](#)
- ## P
- [PERSISTENT_JOB_DATA](#) (pyfarm.jobtypes.core.jobtype.JobType attribute), [47](#)
[pid](#) (pyfarm.jobtypes.core.process.ProcessProtocol attribute), [56](#)
[pop\(\)](#) (pyfarm.agent.config.LoggingConfiguration method), [37](#)
[POP_CONFIG_KEYS](#) (pyfarm.agent.testutil.TestCase attribute), [41](#)
[port\(\)](#) (in module pyfarm.agent.entrypoints.parser), [20](#)
[post\(\)](#) (pyfarm.agent.http.api.assign.Assign method), [22](#)
[post\(\)](#) (pyfarm.agent.http.api.software.CheckSoftware method), [24](#)
[post\(\)](#) (pyfarm.agent.http.api.state.Restart method), [24](#)
[post\(\)](#) (pyfarm.agent.http.api.state.Stop method), [24](#)
[post\(\)](#) (pyfarm.agent.http.api.update.Update method), [26](#)
[post_agent_to_master\(\)](#) (pyfarm.agent.service.Agent method), [39](#)
[post_shutdown_to_master\(\)](#) (pyfarm.agent.service.Agent method), [39](#)
[prepare_config\(\)](#) (pyfarm.agent.testutil.TestCase method), [41](#)
[prepare_for_job\(\)](#) (pyfarm.jobtypes.core.jobtype.JobType class method), [48](#)
[preprocess_stderr_line\(\)](#) (pyfarm.jobtypes.core.jobtype.JobType method), [54](#)
[preprocess_stdout_line\(\)](#) (pyfarm.jobtypes.core.jobtype.JobType method), [54](#)
[Process](#) (class in pyfarm.jobtypes.core.internals), [46](#)
[process](#) (pyfarm.jobtypes.core.process.ProcessProtocol attribute), [56](#)
[process_memory\(\)](#) (in module pyfarm.agent.sysinfo.memory), [33](#)
[process_output\(\)](#) (pyfarm.jobtypes.core.jobtype.JobType method), [52](#)
[PROCESS_PROTOCOL](#) (pyfarm.jobtypes.core.jobtype.JobType attribute), [48](#)
[process_started\(\)](#) (pyfarm.jobtypes.core.jobtype.JobType method), [52](#)
[process_stderr_line\(\)](#) (pyfarm.jobtypes.core.jobtype.JobType method), [55](#)
[process_stdout_line\(\)](#) (pyfarm.jobtypes.core.jobtype.JobType method), [56](#)
[process_stopped\(\)](#) (pyfarm.jobtypes.core.jobtype.JobType method), [52](#)
[ProcessData](#) (class in pyfarm.jobtypes.core.internals), [45](#)
[processEnded\(\)](#) (pyfarm.jobtypes.core.process.ProcessProtocol method), [56](#)
[ProcessProtocol](#) (class in pyfarm.jobtypes.core.process), [56](#)
[protocol](#) (pyfarm.jobtypes.core.internals.ProcessData attribute), [45](#)
[psutil_process](#) (pyfarm.jobtypes.core.process.ProcessProtocol attribute), [56](#)
[putChild\(\)](#) (pyfarm.agent.http.core.resource.Resource method), [30](#)
[putChild\(\)](#) (pyfarm.agent.testutil.APITestServerResource method), [40](#)
[pyfarm.agent](#) (module), [44](#)
[pyfarm.agent.config](#) (module), [36](#)
[pyfarm.agent.entrypoints](#) (module), [21](#)
[pyfarm.agent.entrypoints.development](#) (module), [19](#)
[pyfarm.agent.entrypoints.main](#) (module), [19](#)
[pyfarm.agent.entrypoints.parser](#) (module), [20](#)
[pyfarm.agent.entrypoints.supervisor](#) (module), [21](#)
[pyfarm.agent.entrypoints.utility](#) (module), [21](#)
[pyfarm.agent.http](#) (module), [32](#)
[pyfarm.agent.http.api](#) (module), [26](#)
[pyfarm.agent.http.api.assign](#) (module), [22](#)
[pyfarm.agent.http.api.base](#) (module), [22](#)
[pyfarm.agent.http.api.config](#) (module), [23](#)

pyfarm.agent.http.api.software (module), 23
pyfarm.agent.http.api.state (module), 24
pyfarm.agent.http.api.tasklogs (module), 24
pyfarm.agent.http.api.tasks (module), 25
pyfarm.agent.http.api.update (module), 26
pyfarm.agent.http.core (module), 31
pyfarm.agent.http.core.client (module), 26
pyfarm.agent.http.core.resource (module), 29
pyfarm.agent.http.core.server (module), 30
pyfarm.agent.http.core.template (module), 31
pyfarm.agent.http.system (module), 31
pyfarm.agent.manhole (module), 37
pyfarm.agent.service (module), 38
pyfarm.agent.sysinfo (module), 36
pyfarm.agent.sysinfo.cpu (module), 32
pyfarm.agent.sysinfo.disks (module), 33
pyfarm.agent.sysinfo.graphics (module), 33
pyfarm.agent.sysinfo.memory (module), 33
pyfarm.agent.sysinfo.network (module), 34
pyfarm.agent.sysinfo.software (module), 34
pyfarm.agent.sysinfo.system (module), 35
pyfarm.agent.sysinfo.user (module), 35
pyfarm.agent.testutil (module), 40
pyfarm.agent.utility (module), 42
pyfarm.jobtypes (module), 57
pyfarm.jobtypes.core (module), 57
pyfarm.jobtypes.core.internals (module), 45
pyfarm.jobtypes.core.jobtype (module), 46
pyfarm.jobtypes.core.process (module), 56

Q

queue() (pyfarm.agent.http.core.client.HTTPLog static method), 27
quote_url() (in module pyfarm.agent.utility), 42

R

RAND_LENGTH (pyfarm.agent.testutil.TestCase attribute), 41
random() (in module pyfarm.agent.entrypoints.development), 19
random() (in module pyfarm.agent.http.core.client), 29
random_port() (in module pyfarm.agent.testutil), 40
reannounce() (pyfarm.agent.service.Agent method), 39
REDIRECT_TARGET (pyfarm.agent.testutil.BaseRequestTestCase attribute), 41
register_callback() (pyfarm.agent.config.ConfigurationWithCallbacks class method), 37
remove_directory() (in module pyfarm.agent.utility), 43
remove_file() (in module pyfarm.agent.utility), 43
render() (pyfarm.agent.http.core.resource.Resource method), 30

render() (pyfarm.agent.http.core.server.StaticPath method), 30
render_deferred() (pyfarm.agent.http.core.resource.Resource method), 30
render_DELETE() (pyfarm.agent.testutil.APITestServerResource method), 40
render_GET() (pyfarm.agent.testutil.APITestServerResource method), 40
render_POST() (pyfarm.agent.testutil.APITestServerResource method), 40
render_PUT() (pyfarm.agent.testutil.APITestServerResource method), 40
render_tuple() (pyfarm.agent.http.core.resource.Resource method), 30
repeating_call() (pyfarm.agent.service.Agent method), 38
replace_list() (pyfarm.agent.testutil.TestCase method), 41
ReplaceEnvironment (class in pyfarm.jobtypes.core.process), 56
Request (class in pyfarm.agent.http.core.client), 27
request() (in module pyfarm.agent.http.core.client), 28
request() (pyfarm.agent.testutil.BaseHTTPTestCase method), 42
request_from_master() (in module pyfarm.agent.utility), 42
requestAvatar() (pyfarm.agent.manhole.TelnetRealm method), 38
Resource (class in pyfarm.agent.http.core.resource), 29
Response (class in pyfarm.agent.http.core.client), 28
response() (pyfarm.agent.http.core.client.HTTPLog static method), 27
Restart (class in pyfarm.agent.http.api.state), 24
retry() (pyfarm.agent.http.core.client.Request method), 27
RFC
RFC 1918, 34
running() (pyfarm.jobtypes.core.process.ProcessProtocol method), 57

S

save() (pyfarm.agent.utility.AgentUUID class method), 43
SCHEMAS (pyfarm.agent.http.api.assign.Assign attribute), 22
SCHEMAS (pyfarm.agent.http.api.software.CheckSoftware attribute), 23
SCHEMAS (pyfarm.agent.http.api.state.Stop attribute), 24
SCHEMAS (pyfarm.agent.http.api.update.Update attribute), 26
SCHEMAS (pyfarm.agent.http.core.resource.Resource attribute), 29
seconds() (in module pyfarm.agent.http.system), 31

set_content() (pyfarm.agent.testutil.DummyRequest method), 40
 set_default_environment() (pyfarm.jobtypes.core.jobtype.CommandData method), 47
 set_response_code_if_not_set() (pyfarm.agent.http.core.resource.Resource method), 30
 set_states() (pyfarm.jobtypes.core.jobtype.JobType method), 51
 set_task_progress() (pyfarm.jobtypes.core.jobtype.JobType method), 51
 set_task_started_now() (pyfarm.jobtypes.core.jobtype.JobType method), 51
 set_task_state() (pyfarm.jobtypes.core.jobtype.JobType method), 52
 setHeader() (pyfarm.agent.testutil.DummyRequest method), 41
 setUp() (pyfarm.agent.testutil.BaseHTTPTestCase method), 42
 setUp() (pyfarm.agent.testutil.BaseRequestTestCase method), 41
 setUp() (pyfarm.agent.testutil.TestCase method), 41
 should_reannounce() (pyfarm.agent.service.Agent method), 39
 show() (in module pyfarm.agent.manhole), 38
 shutting_down (pyfarm.agent.service.Agent attribute), 38
 sigint_handler() (pyfarm.agent.service.Agent method), 39
 Site (class in pyfarm.agent.http.core.server), 30
 size (pyfarm.agent.sysinfo.disks.DiskInfo attribute), 33
 skipIf (class in pyfarm.agent.testutil), 40
 spawn_persistent_process() (pyfarm.jobtypes.core.jobtype.JobType class method), 48
 start() (pyfarm.agent.entrypoints.main.AgentEntryPoint method), 19
 start() (pyfarm.agent.service.Agent method), 39
 start() (pyfarm.jobtypes.core.jobtype.JobType method), 51
 start_daemon_posix() (in module pyfarm.agent.entrypoints.utility), 21
 started (pyfarm.jobtypes.core.internals.ProcessData attribute), 45
 StaticPath (class in pyfarm.agent.http.core.server), 30
 Status (class in pyfarm.agent.http.api.state), 24
 status() (pyfarm.agent.entrypoints.main.AgentEntryPoint method), 20
 Stop (class in pyfarm.agent.http.api.state), 24
 stop() (pyfarm.agent.entrypoints.main.AgentEntryPoint method), 20
 stop() (pyfarm.agent.service.Agent method), 39
 stop() (pyfarm.agent.testutil.FakeAgent method), 40
 stop() (pyfarm.jobtypes.core.jobtype.JobType method), 51
 stopped (pyfarm.jobtypes.core.internals.ProcessData attribute), 45
 StoreAction (in module pyfarm.agent.entrypoints.parser), 20
 StoreConstAction (in module pyfarm.agent.entrypoints.parser), 20
 StoreFalseAction (in module pyfarm.agent.entrypoints.parser), 21
 StoreTrueAction (in module pyfarm.agent.entrypoints.parser), 21
 SubParsersAction (in module pyfarm.agent.entrypoints.parser), 20
 supervisor() (in module pyfarm.agent.entrypoints.supervisor), 21
 System (class in pyfarm.jobtypes.core.internals), 46
 system_data() (pyfarm.agent.service.Agent method), 39
 system_time() (in module pyfarm.agent.sysinfo.cpu), 32

T

TaskLogs (class in pyfarm.agent.http.api.tasklogs), 24
 TaskNotFound, 46
 Tasks (class in pyfarm.agent.http.api.tasks), 25
 TASKS_SCHEMA() (in module pyfarm.agent.utility), 42
 TelnetRealm (class in pyfarm.agent.manhole), 38
 tempdir() (pyfarm.jobtypes.core.jobtype.JobType method), 49
 TEMPLATE (pyfarm.agent.http.core.resource.Resource attribute), 29
 template (pyfarm.agent.http.core.resource.Resource attribute), 30
 TEMPLATE (pyfarm.agent.http.system.Configuration attribute), 32
 TEMPLATE (pyfarm.agent.http.system.Index attribute), 31
 terminate() (pyfarm.jobtypes.core.process.ProcessProtocol method), 57
 test_implements_methods() (pyfarm.agent.testutil.BaseHTTPTestCase method), 42
 test_instance() (pyfarm.agent.testutil.BaseHTTPTestCase method), 42
 test_leaf() (pyfarm.agent.testutil.BaseHTTPTestCase method), 42
 test_methods_exist_for_schema() (pyfarm.agent.testutil.BaseHTTPTestCase method), 42
 test_parent() (pyfarm.agent.testutil.BaseAPITestCase method), 42
 test_template_loaded() (pyfarm.agent.testutil.BaseHTMLTestCase method), 42

test_template_set() (pyfarm.agent.testutil.BaseHTMLTestCase method), 42

TEST_URL (pyfarm.agent.testutil.BaseRequestTestCase attribute), 41

TestCase (class in pyfarm.agent.testutil), 41

TestCaseLogHandler (class in pyfarm.agent.testutil), 41

testing (pyfarm.agent.http.api.software.CheckSoftware attribute), 23

TimedDeferredLock (class in pyfarm.agent.utility), 44

timeout (pyfarm.agent.testutil.TestCase attribute), 41

total_consumption() (in module pyfarm.agent.sysinfo.memory), 33

total_cpus() (in module pyfarm.agent.sysinfo.cpu), 32

total_ram() (in module pyfarm.agent.sysinfo.memory), 33

total_seconds() (in module pyfarm.agent.utility), 43

TransportProtocolFactory (class in pyfarm.agent.manhole), 38

TYPE_MAPPING (pyfarm.agent.entrypoints.parser.ActionMixin attribute), 20

TypeChecks (class in pyfarm.jobtypes.core.internals), 46

U

uidgid() (in module pyfarm.agent.entrypoints.parser), 20

UnicodeCSVReader (class in pyfarm.agent.utility), 43

UnicodeCSVWriter (class in pyfarm.agent.utility), 43

Update (class in pyfarm.agent.http.api.update), 26

update() (pyfarm.agent.config.LoggingConfiguration method), 37

uptime() (in module pyfarm.agent.sysinfo.system), 35

URI (pyfarm.agent.testutil.BaseHTTPTestCase attribute), 41

used_ram() (in module pyfarm.agent.sysinfo.memory), 33

user_time() (in module pyfarm.agent.sysinfo.cpu), 32

username() (in module pyfarm.agent.sysinfo.user), 35

UTF8Recoder (class in pyfarm.agent.utility), 43

uuid (pyfarm.jobtypes.core.internals.System attribute), 46

uuid (pyfarm.jobtypes.core.process.ProcessProtocol attribute), 56

uuid_type() (in module pyfarm.agent.entrypoints.parser), 20

V

validate() (pyfarm.jobtypes.core.jobtype.CommandData method), 47

validate_environment() (in module pyfarm.agent.utility), 42

validate_uuid() (in module pyfarm.agent.utility), 42

VersionNotFound, 34

Versions (class in pyfarm.agent.http.api.base), 22

W

write() (pyfarm.agent.testutil.DummyRequest method), 41

writerow() (pyfarm.agent.utility.UnicodeCSVWriter method), 43

writerows() (pyfarm.agent.utility.UnicodeCSVWriter method), 43