
pyfarm.agent Documentation

Release 0.8.2

Oliver Palmer, Guido Winkelmann

February 17, 2015

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	16
4	pyfarm.agent package	19
4.1	Subpackages	19
4.2	Submodules	31
4.3	Module contents	39
5	pyfarm.jobtypes package	41
5.1	Subpackages	41
5.2	Submodules	52
5.3	Module contents	52
6	Indices and tables	53
	HTTP Routing Table	55
	Python Module Index	57

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

Commands

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]
[--projects PROJECTS [PROJECTS ...]]
[--state STATE]
[--time-offset TIME_OFFSET]
[--ntp-server NTP_SERVER]
[--ntp-server-version NTP_SERVER_VERSION]
[--no-pretty-json]
[--shutdown-timeout SHUTDOWN_TIMEOUT]
[--updates-drop-dir UPDATES_DROP_DIR]
[--run-control-file RUN_CONTROL_FILE]
[--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 HTTP_RETRY_DELAY]
[--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.

```

--projects PROJECTS [PROJECTS ...]
    The project or projects this agent is dedicated to. By
    default the agent will service any project however
    specific projects may be specified. For example if you
    wish this agent to service 'Foo Part I' and 'Foo Part
    II' only just specify it as '--projects "Foo Part I"
    "Foo Part II"'
--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]
```

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 HTTP_RETRY_DELAY

If a http request to the master has failed, wait this amount of time before trying again

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.

`--segsfault` 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:

<code>-h, --help</code>	show this help message and exit
<code>--master-api MASTER_API</code>	The url to the master's api [default: http://127.0.0.1/api/v1]
<code>--agent-api AGENT_API</code>	The url to the agent's api [default: http://127.0.0.1:50000/api/v1]
<code>--jobtype JOBTYP</code>	The job type to use [default: FakeRender]
<code>--job JOB</code>	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`.

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 platform specific locations where the agent uuid
2  # file is default. This can be overridden with --agent-id-file flag.
3  agent_id_file_platform_defaults:
4      linux: /etc/pyfarm/agent/uuid.dat
5      mac: /Library/pyfarm/agent/uuid.dat
6      bsd: /etc/pyfarm/agent/uuid.dat
7      windows: $LOCALAPPDATA/pyfarm/agent/uuid.dat
8
9  # The platform specific locations where the run control file is looked for
10 run_control_file_by_platform:
11     linux: /tmp/pyfarm/agent/should_be_running
12     mac: /tmp/pyfarm/agent/should_be_running
13     bsd: /tmp/pyfarm/agent/should_be_running
14     windows: $TEMP/pyfarm/agent/should_be_running
15
16 # The default location to store data. $temp will expand to
17 # whatever pyfarm's data root is plus the application
18 # name (agent). For example on Linux this would expand to
19 # /tmp/pyfarm/agent
20 agent_data_root: $temp
21
22 # Defines the number of seconds between iterations of pyfarm-supervisor's
23 # agent status check.
24 supervisor_interval: 5
25
26 # The location where the agent should change directories
27 # into upon starting. If this value is not set then no
28 # changes will be made.
29 agent_chdir:
30
31 # The location where static web files should be served from. This

```

```
32 # will default to using PyFarm's installation root.
33 agent_static_root: auto
34
35 # The default location where lock files should be stored. By
36 # default these will be stored alone side other data
37 # inside the 'agent_data_root' value above.
38 lock_file_root: $agent_data_root/lock
39
40 # Locations of specific lock files
41 agent_lock_file: $lock_file_root/agent.pid
42 supervisor_lock_file: $lock_file_root/supervisor.pid
43
44 # Where user data for the agent is stored. ~ will be expanded
45 # to the current users's home directory.
46 agent_user_data: ~/.pyfarm/agent
47
48 # The default location where the agent should save logs to. This
49 # includes both logs from processes and the agent log itself.
50 agent_logs_root: $agent_data_root/logs
51
52 # The location where agent updates should be stored.
53 agent_updates_dir: $agent_data_root/updates
54
55 # The default port which the agent should use to serve the
56 # REST api.
57 agent_api_port: 50000
58
59 # The location where the the agent should save its own
60 # logging output to.
61 agent_log: $agent_logs_root/agent.log
62 supervisor_log: $agent_logs_root/supervisor.log
63
64 # The user agent the master will use when connecting to the agent's
65 # REST api. This value should only be changed if the master's code
66 # is updated with a new user agent. Change this value has not effect
67 # on the master.
68 master_user_agent: PyFarm/1.0 (master)
69
70 # Configuration values which control how the url
71 # for the master is constructed. If 'master' is not set
72 # the --master flag will be required to start the agent.
73 master:
74 master_api_version: 1
75 master_api: http://$master/api/v$master_api_version
76
77 # The user agent the master uses to talke to the agent's
78 # REST api. This value should not be modified unless
79 # there's a specific reason to do so.
80 master_user_agent: PyFarm/1.0 (master)
81
82 # Controls how often the agent should reannounce itself
83 # to the master. The agent may be in contact with the master
84 # more often than this however during long period of
85 # inactivity this is how often the agent will 'inform' the
86 # master the agent is still online.
87 agent_master_reannounce: 120
88
89 # How many seconds the agent should spend attempting to inform
```

```

90  # the master that it's shutting down.
91  agent_shutdown_timeout: 15
92
93  # If an http request fails, use this as the base value
94  # to help determine how long we should wait before retrying
95  agent_http_retry_delay: 5
96
97  # Controls if the http client connection should be persistent or
98  # not. Generally this should always be True because the connection
99  # self-terminates after a short period of time anyway. For higher
100 # latency situations or with larger deployments this value should
101 # be False.
102 agent_http_persistent_connections: True
103
104 # If True then html templates will be reloaded with
105 # every request instead of cached.
106 agent_html_template_reload: False
107
108 # If True then reformat json output to be more human
109 # readable.
110 agent_pretty_json: True
111
112 # How often the agent should check for changes in ram. This value
113 # is used to ensure ram usage is checked at least this often though
114 # it may be checked more often due to other events (such as jobs
115 # running)
116 agent_ram_check_interval: 30
117
118 # If the ram has changed this may megabytes since the last
119 # check then report the change to the master.
120 agent_ram_report_delta: 100
121
122 # How much the agent should wait, in seconds, between
123 # each report about a change in ram.
124 agent_ram_max_report_frequency: 10
125
126 # The default network time server and version the agent
127 # should use to calculate its clock skew.
128 agent_ntp_server: pool.ntp.org
129 agent_ntp_server_version: 2
130
131 # The amount of time this agent is offset from what
132 # would be considered correct based on an atomic
133 # clock. If this value is set to auto the time will
134 # be calculated using NTP.
135 agent_time_offset: auto
136
137 # Physical and network information about the host the agent
138 # is running on. Setting these values to 'auto' will cause
139 # them to be initialized to the system's current
140 # configuration values.
141 agent_ram: auto
142 agent_cpus: auto
143 agent_hostname: auto
144
145 # When True this will enable a telnet connection
146 # to the agent which will present a Python interpreter
147 # upon connection. This is mainly used for debugging

```

```
148 # and direct manipulation of the agent. You can use
149 # the show() function once connected to see what
150 # objects are available.
151 agent_manhole: False
152 agent_manhole_port: 50001
153 agent_manhole_username: admin
154 agent_manhole_password: admin
155
156 # NOTE: The following values are used by the unittests and should be
157 # generally ignored for anything other than development.
158 agent_unittest:
159     dns_test_hostname: example.com
160     client_redirect_target: http://example.com
161     client_api_test_url_https: https://httpbin.pyfarm.net
162     client_api_test_url_http: http://httpbin.pyfarm.net
163
164 # A list of paths or names where the 'lspci' command can
165 # be called from on Linux. This is used to retrieve information
166 # about graphics cards installed on the system in
167 # 'pyfarm.agent.sysinfo.graphics.graphics_cards'.
168 # If you need run the command with sudo you may also specify an entry
169 # like this:
170 # - sudo lspci
171 sysinfo_command_lspci:
172     - lspci
173     - /bin/lspci
174     - /sbin/lspci
175     - /usr/sbin/lspci
176     - /usr/bin/lspci
```

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 # $HOURL - 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
45 # If True, then ignore any errors produced when trying
46 # to map users and groups to IDs. This will cause the
47 # underlying methods in the job type to instead run
48 # as the job type's owner instead, ignoring what the
49 # incoming job requests.
50 # NOTE: This value is not used on Windows.
51 jobtype_ignore_id_mapping_errors: False
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
59 # Configures the thread pool used by job types
60 # for logging.
61 jobtype_logging_threadpool:
62     # Setting this value to something smaller than '1' will result
63     # in an exception being raised. This value also cannot be larger
64     # than 'max_threads' below.
65     min_threads: 3
66
67     # This value must be greater than or equal to 'min_threads'
68     # above. You may also set this value to 'auto' meaning the
69     # number of processors times 1.5 or 20 (whichever is lower).
70     max_threads: auto
71
72     # As log messages are sent from processes they are stored
73     # in an in memory queue. When the number of messages is higher
74     # than this number a thread will be spawned to consume the
75     # data and flush it into a file object.
76     max_queue_size: 10
77
78     # Most often the operating system will control how often data
79     # is written to disk from a file object. This value overrides

```

```
80     # that behavior and forces the file object to flush to disk
81     # after this many messages have been processed.
82     flush_lines: 100
```

pyfarm.agent package

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 at 0x7fa5d255a0c8>: <function direxists at 0x7fa5cb2f9cf8>, <function isfile at 0x7fa5

`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': <voluptuous.Schema object at 0x7fa5cab0d110>}
    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
    CONTENT_TYPES = set(['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.state module

```
class pyfarm.agent.http.api.state.Stop
    Bases: pyfarm.agent.http.api.base.APIResource

    isLeaf = False

    SCHEMAS = {'POST': <voluptuous.Schema object at 0x7fa5ca89cad0>}

    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
```

pyfarm.agent.http.api.tasks module

```
class pyfarm.agent.http.api.tasks.Tasks
    Bases: pyfarm.agent.http.api.base.APIResource

    get (**kwargs)

    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.
```

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': <voluptuous.Schema object at 0x7fa5ca78a950>}

    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 (initial=None, uniform=False, get_delay=<built-in method random of Random object at 0x167d2b0>, minimum=1)`

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. This helps to ensure that if the same request, such as agents coming online, is being run on multiple systems they should be staggered a little more than they would be without the non-uniform delay.

Parameters

- **initial** (*int*) – The initial delay value to start off with before any extra calculations are done. If this value is not provided the value provided to `--http-retry-delay` at startup will be used.
- **uniform** (*bool*) – If True then use the value produced by `get_delay` as a multiplier.
- **get_delay** (*callable*) – A function which should produce a number to multiply delay by. By default this uses `random.random()`

- **minimum** – Ensures that the value returned from this function is greater than or equal to a minimum value.

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 id called before all data has been received.

json (*loader=<function loads at 0x7fa5cee8d488>*)

Returns the json data from the incoming request

Raises

- **RuntimeError** – Raised if this method id 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 <class 'twisted.internet.error.ConnectionDone'>>*)

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:

- rewriting of request objects
- templating
- content type discovery and validation
- unpacking of request data
- rerouting of request to specific internal methods

TEMPLATE = `NotImplemented`

CONTENT_TYPES = `set(['application/json', 'text/html'])`

LOAD_DATA_FOR_METHODS = `set(['PUT', 'POST'])`

SCHEMAS = `{}`

template

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

methods

A set containing all the methods this resource implements.

content_types (*request*, *default=None*)

Returns the content type(s) present in the request

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

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.RewriteRequest(*args, **kw)
    Bases: twisted.web.server.Request

    A custom implementation of _Request that will allow us to modify an incoming request before it reaches the
    HTTP server.

    REPLACE_REPEATED_DELIMITER = <_sre.SRE_Pattern object at 0x7fa5cafca270>

    requestReceived(command, path, version)
        Override the built in _Request.requestReceived() so we can rewrite portions of the request, such
        as the url, before it's passed along to the internal server.

    write(data)
        Override the built in _Request.write() so that any data that's not a string will be dumped to json
        using dumps()
```

```
class pyfarm.agent.http.core.server.Site(resource, *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

    requestFactory
        alias of RewriteRequest
```

```
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.DeferredTemplate
    Bases: jinja2.environment.Template

    Overrides the default PackageLoader so we can produced the rendered result as a deferred call.

    render (*args, **kwargs)

class pyfarm.agent.http.core.template.Environment (**kwargs)
    Bases: jinja2.environment.Environment

    Implementation of Jinja's _Environment class which reads from our configuration object and establishes the
    default functions we can use in a template.

    template_class
        alias of DeferredTemplate

class pyfarm.agent.http.core.template.Loader
    Bases: object

    Namespace class used to simply keep track of the global environment and load templates.

    >>> from pyfarm.agent.http.core import template
    >>> template.load("index.html")

    environment = None

    classmethod load (name)
```

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

serves request for the root, '/', target

```
TEMPLATE = 'index.html'
```

```
get (**kwargs)
```

```
class pyfarm.agent.http.system.Configuration
    Bases: pyfarm.agent.http.core.resource.Resource
```

```
TEMPLATE = 'configuration.html'
```

```
HIDDEN_FIELDS = ('agent', 'agent_pretty_json')
```

```
EDITABLE_FIELDS = ('agent_cpus', 'agent_hostname', 'agent_http_retry_delay', 'master_api', 'master', 'agent_ram_c
```

```
get (**kwargs)
```


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.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.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 at 0x7fa5d24e8480>, *old_value*=<object object at 0x7fa5d24e8480>)

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 at 0x7fa5d24e8480>, *old_value*=<object object at 0x7fa5d24e8480>)

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

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 at 0x7fa5d24e8480>*)

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 ()

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

system_data (*requery_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 ()

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 (*stop_reactor=True*)

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_post_free_ram (*response*)

Called when we get a response back from the master after POSTing a change for *free_ram*

errback_post_free_ram (*failure*)

Error handler which is called if we fail to post a ram update to the master for some reason

post_free_ram ()

Posts the current nu

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

Callback used to decide and act on changes to the `config['ram']` value.

errback_post_cpu_count_change (*failure*)

Error handler which is called if we fail to post a cpu count update to an existing agent for some reason.

callback_post_cpu_count_change (*response*)

Called when we received a response from the master after

post_cpu_count (*run=True*)

POSTs CPU count changes to the master

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

Callback used to decide and act on changes to the `config['cpus']` value.

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

4.2.4 pyfarm.agent.tasks module

4.2.5 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.requires_master` (*function*)
Any test decorated with this function will fail if the master could not be contacted or returned a response other than 200 OK for “/”

`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.FakeRequest` (*test, method, uri, headers=None, data=None*)
Bases: `object`

`getHeader` (*header*)
`setResponseCode` (*code*)
`write` (*data*)
`finish` ()
`response` ()

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.TestCase` (*methodName='runTest'*)
Bases: `twisted.trial._async_test.TestCase`

`longMessage` = `True`
`POP_CONFIG_KEYS` = []
`RAND_LENGTH` = 8
`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*)
`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 manage 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'

    DNS_HOSTNAME = 'example.com'

    TEST_URL = 'http://httpbin.pyfarm.net'

    REDIRECT_TARGET = 'http://example.com'

    RESOLVED_DNS_NAME = True

    HTTP_REQUEST_SUCCESS = True

    setUp ()

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

    URI = NotImplemented

    CLASS = NotImplemented

    CLASS_FACTORY = NotImplemented

    CONTENT_TYPES = NotImplemented

    setUp ()

    instance_class ()

    test_instance ()

    test_leaf ()

    test_implements_methods ()

    test_content_types ()

    test_methods_exist_for_schema ()

    test_missing_schemas ()

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

    CONTENT_TYPES = ['application/json']

    setUp ()

    test_parent ()

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

    CONTENT_TYPES = ['text/html', 'application/json']

    setUp ()

    test_template_set ()

```

```
test_template_loaded()
```

4.2.6 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 at 0x7fa5cc9ebf58>,
                                             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 at 0x7fa5cc9ebf58>,
                                             encoding='utf-8', **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 at 0x7fa5cc762a10>

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.

4.3 Module contents

4.3.1 PyFarm Agent

Core module containing code to run PyFarm's agent.

pyfarm.jobtypes package

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`

Bases: `tuple`

`ProcessData(protocol, started, stopped, log_identifier)`

log_identifier

Alias for field number 3

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 = {}

JOBTYPE_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 = {}
```

```
    stopped_deferred
```

```
    start_deferred
```

```
class pyfarm.jobtypes.core.internals.System
```

```
    Bases: object
```

```
class pyfarm.jobtypes.core.internals.TypeChecks
```

```
    Bases: object
```

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.

```
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 = <voluptuous.Schema object at 0x7fa5cab00cd0>

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 similar 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 (*cmdlist*)

Return a list of command to be used when running the process as a read-only tuple.

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.

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.

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

Expands variables inside of a string using an environment. Exp

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*, *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.
- **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. If we don't know how to format the request object an error will be logged and nothing will be returned

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_state (*task*, *state*, *error=None*, *dissociate_agent=False*)

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 (*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 if 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_stderr_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 Submodules

5.2.1 pyfarm.jobtypes.examples module

```
class pyfarm.jobtypes.examples.PythonHelloWorld(assignment)  
    Bases: pyfarm.jobtypes.core.jobtype.JobType  
    get_command_data()
```

5.3 Module contents

5.3.1 Job Types

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

Indices and tables

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

/api

GET /api/v1/versions/ HTTP/1.1, [22](#)
POST /api/v1/update HTTP/1.1, [23](#)

p

- `pyfarm.agent`, 39
- `pyfarm.agent.config`, 31
- `pyfarm.agent.entrypoints`, 21
- `pyfarm.agent.entrypoints.development`, 19
- `pyfarm.agent.entrypoints.main`, 19
- `pyfarm.agent.entrypoints.parser`, 19
- `pyfarm.agent.entrypoints.supervisor`, 21
- `pyfarm.agent.entrypoints.utility`, 21
- `pyfarm.agent.http`, 29
 - `pyfarm.agent.http.api`, 23
 - `pyfarm.agent.http.api.assign`, 21
 - `pyfarm.agent.http.api.base`, 21
 - `pyfarm.agent.http.api.config`, 22
 - `pyfarm.agent.http.api.state`, 22
 - `pyfarm.agent.http.api.tasklogs`, 23
 - `pyfarm.agent.http.api.tasks`, 23
 - `pyfarm.agent.http.api.update`, 23
 - `pyfarm.agent.http.core`, 28
 - `pyfarm.agent.http.core.client`, 24
 - `pyfarm.agent.http.core.resource`, 26
 - `pyfarm.agent.http.core.server`, 26
 - `pyfarm.agent.http.core.template`, 27
 - `pyfarm.agent.http.system`, 28
- `pyfarm.agent.manhole`, 33
- `pyfarm.agent.service`, 34
- `pyfarm.agent.sysinfo`, 31
 - `pyfarm.agent.sysinfo.cpu`, 29
 - `pyfarm.agent.sysinfo.graphics`, 29
 - `pyfarm.agent.sysinfo.memory`, 30
 - `pyfarm.agent.sysinfo.network`, 30
 - `pyfarm.agent.sysinfo.system`, 31
 - `pyfarm.agent.sysinfo.user`, 31
- `pyfarm.agent.testutil`, 36
- `pyfarm.agent.utility`, 38
- `pyfarm.jobtypes`, 52
 - `pyfarm.jobtypes.core`, 52
 - `pyfarm.jobtypes.core.process`, 51
 - `pyfarm.jobtypes.core.examples`, 52
 - `pyfarm.jobtypes.core.internals`, 41
 - `pyfarm.jobtypes.core.jobtype`, 42

Symbols

`_expandvars()` (pyfarm.agent.config.LoggingConfiguration method), 32

A

ActionMixin (class in pyfarm.agent.entrypoints.parser), 20
 addresses() (in module pyfarm.agent.sysinfo.network), 30
 Agent (class in pyfarm.agent.service), 34
 agent_api (pyfarm.agent.entrypoints.main.AgentEntryPoint attribute), 19
 agent_api() (pyfarm.agent.service.Agent class method), 34
 AgentArgumentParser (class in pyfarm.agent.entrypoints.parser), 20
 AgentEntryPoint (class in pyfarm.agent.entrypoints.main), 19
 agents_endpoint() (pyfarm.agent.service.Agent class method), 34
 AgentUUID (class in pyfarm.agent.utility), 38
 ALLOW_DIRECTORY_LISTING (pyfarm.agent.http.core.server.StaticPath attribute), 27
 APIResource (class in pyfarm.agent.http.api.base), 21
 APIRoot (class in pyfarm.agent.http.api.base), 21
 AppendAction (in module pyfarm.agent.entrypoints.parser), 20
 AppendConstAction (in module pyfarm.agent.entrypoints.parser), 20
 assert_parser() (in module pyfarm.agent.entrypoints.parser), 19
 assertDateAlmostEqual() (pyfarm.agent.testutil.TestCase method), 36
 assertRaises() (pyfarm.agent.testutil.TestCase method), 36
 assertRaisesRegexp() (pyfarm.agent.testutil.TestCase method), 36
 Assign (class in pyfarm.agent.http.api.assign), 21
 ASSIGNMENT_SCHEMA (pyfarm.jobtypes.core.jobtype.JobType attribute),

43

assignments() (pyfarm.jobtypes.core.jobtype.JobType method), 45

B

BaseAPITestCase (class in pyfarm.agent.testutil), 37
 BaseHTMLTestCase (class in pyfarm.agent.testutil), 37
 BaseHTTPTestCase (class in pyfarm.agent.testutil), 37
 BaseRequestTestCase (class in pyfarm.agent.testutil), 37
 before_spawn_process() (pyfarm.jobtypes.core.jobtype.JobType method), 47
 before_start() (pyfarm.jobtypes.core.jobtype.JobType method), 47
 build_http_resource() (pyfarm.agent.service.Agent method), 35
 build_url() (in module pyfarm.agent.http.core.client), 24

C

Cache (class in pyfarm.jobtypes.core.internals), 41
 cache (pyfarm.agent.http.core.template.InMemoryCache attribute), 27
 cache (pyfarm.jobtypes.core.internals.Cache attribute), 41
 CACHE_DIRECTORY (pyfarm.jobtypes.core.internals.Cache attribute), 41
 callback_agent_id_set() (pyfarm.agent.service.Agent method), 35
 callback_cpu_count_changed() (pyfarm.agent.service.Agent method), 35
 callback_free_ram_changed() (pyfarm.agent.service.Agent method), 35
 callback_post_cpu_count_change() (pyfarm.agent.service.Agent method), 35
 callback_post_free_ram() (pyfarm.agent.service.Agent method), 35
 callbacks (pyfarm.agent.config.ConfigurationWithCallbacks attribute), 33
 changed() (pyfarm.agent.config.ConfigurationWithCallbacks method), 33

- changed() (pyfarm.agent.config.LoggingConfiguration method), 32
- CLASS (pyfarm.agent.testutil.BaseHTTPTestCase attribute), 37
- CLASS_FACTORY (pyfarm.agent.testutil.BaseHTTPTestCase attribute), 37
- cleanup_after_job() (pyfarm.jobtypes.core.jobtype.JobType class method), 44
- clear() (pyfarm.agent.config.ConfigurationWithCallbacks method), 33
- clear() (pyfarm.agent.config.LoggingConfiguration method), 32
- clear() (pyfarm.agent.http.core.template.InMemoryCache method), 27
- COMMAND_DATA (pyfarm.jobtypes.core.jobtype.JobType attribute), 43
- CommandData (class in pyfarm.jobtypes.core.jobtype), 42
- Config (class in pyfarm.agent.http.api.config), 22
- Configuration (class in pyfarm.agent.http.system), 28
- ConfigurationWithCallbacks (class in pyfarm.agent.config), 32
- connectionLost() (pyfarm.agent.http.core.client.Response method), 25
- connectionLost() (pyfarm.agent.manhole.LoggingManhole method), 33
- connectionMade() (pyfarm.agent.manhole.LoggingManhole method), 33
- connectionMade() (pyfarm.jobtypes.core.process.ProcessProtocol method), 51
- CONTENT_TYPES (pyfarm.agent.http.api.base.APIResource attribute), 21
- CONTENT_TYPES (pyfarm.agent.http.core.resource.Resource attribute), 26
- CONTENT_TYPES (pyfarm.agent.testutil.BaseAPITestCase attribute), 37
- CONTENT_TYPES (pyfarm.agent.testutil.BaseHTMLTestCase attribute), 37
- CONTENT_TYPES (pyfarm.agent.testutil.BaseHTTPTestCase attribute), 37
- content_types() (pyfarm.agent.http.core.resource.Resource method), 26
- cpu_name() (in module pyfarm.agent.sysinfo.cpu), 29
- create_directory() (pyfarm.agent.testutil.TestCase method), 37
- create_file() (pyfarm.agent.testutil.TestCase method), 36
- create_jobtype() (in module pyfarm.agent.testutil), 36
- CREATED (pyfarm.agent.config.LoggingConfiguration attribute), 32
- ## D
- data() (pyfarm.agent.http.core.client.Response method), 25
- dataReceived() (pyfarm.agent.http.core.client.Response method), 25
- default_json_encoder() (in module pyfarm.agent.utility), 38
- DeferredTemplate (class in pyfarm.agent.http.core.template), 27
- delete() (pyfarm.agent.http.api.tasks.Tasks method), 23
- DELETED (pyfarm.agent.config.LoggingConfiguration attribute), 32
- deregister_callback() (pyfarm.agent.config.ConfigurationWithCallbacks class method), 33
- directoryListing() (pyfarm.agent.http.core.server.StaticPath method), 27
- direxists() (in module pyfarm.agent.entrypoints.parser), 20
- displayTracebacks (pyfarm.agent.http.core.server.Site attribute), 27
- DNS_HOSTNAME (pyfarm.agent.testutil.BaseRequestTestCase attribute), 37
- dump_bytecode() (pyfarm.agent.http.core.template.InMemoryCache method), 27
- dumps() (in module pyfarm.agent.utility), 38
- ## E
- e (pyfarm.jobtypes.core.internals.Cache attribute), 41
- EDITABLE_FIELDS (pyfarm.agent.http.system.Configuration attribute), 28
- enum() (in module pyfarm.agent.entrypoints.parser), 20
- Environment (class in pyfarm.agent.http.core.template), 28
- environment (pyfarm.agent.http.core.template.Loader attribute), 28
- environment variable
- PYFARM_JOBTYPE_ALLOW_CODE_EXECUTION_IN_MODULE, 11
- PYFARM_JOBTYPE_SUBCLASSES_BASE_CLASS, 11
- environment_is_case_sensitive() (in module pyfarm.agent.sysinfo.system), 31
- errback_post_cpu_count_change() (pyfarm.agent.service.Agent method), 35
- errback_post_free_ram() (pyfarm.agent.service.Agent method), 35

- error() (pyfarm.agent.http.core.client.HTTPLog static method), 24
- error() (pyfarm.agent.http.core.resource.Resource method), 26
- error() (pyfarm.agent.testutil.ErrorCapturingParser method), 36
- ErrorCapturingParser (class in pyfarm.agent.testutil), 36
- errReceived() (pyfarm.jobtypes.core.process.ProcessProtocol method), 51
- expandvars() (pyfarm.jobtypes.core.jobtype.JobType method), 46
- EXPIRES (pyfarm.agent.http.core.server.StaticPath attribute), 27
- ## 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), 36
- FakeRequest (class in pyfarm.agent.testutil), 36
- FakeRequestHeaders (class in pyfarm.agent.testutil), 36
- fileexists() (in module pyfarm.agent.entrypoints.parser), 20
- filesystem_is_case_sensitive() (in module pyfarm.agent.sysinfo.system), 31
- finish() (pyfarm.agent.testutil.FakeRequest method), 36
- format_error() (pyfarm.jobtypes.core.jobtype.JobType method), 46
- format_stderr_line() (pyfarm.jobtypes.core.jobtype.JobType method), 49
- format_stdout_line() (pyfarm.jobtypes.core.jobtype.JobType method), 49
- free_ram() (in module pyfarm.agent.sysinfo.memory), 30
- ## G
- generate() (pyfarm.agent.utility.AgentUUID class method), 39
- get() (pyfarm.agent.http.api.base.Versions method), 22
- get() (pyfarm.agent.http.api.config.Config method), 22
- get() (pyfarm.agent.http.api.state.Status method), 22
- get() (pyfarm.agent.http.api.tasklogs.TaskLogs method), 23
- get() (pyfarm.agent.http.api.tasks.Tasks method), 23
- get() (pyfarm.agent.http.system.Configuration method), 28
- get() (pyfarm.agent.http.system.Index method), 28
- get_command_list() (pyfarm.jobtypes.core.jobtype.JobType method), 45
- get_command_data() (pyfarm.jobtypes.examples.PythonHelloWorld method), 52
- get_csvlog_path() (pyfarm.jobtypes.core.jobtype.JobType method), 45
- get_environment() (pyfarm.jobtypes.core.jobtype.JobType method), 45
- get_local_task_state() (pyfarm.jobtypes.core.jobtype.JobType method), 47
- get_uid_gid() (pyfarm.jobtypes.core.jobtype.JobType method), 45
- getHeader() (pyfarm.agent.testutil.FakeRequest method), 36
- getRawHeaders() (pyfarm.agent.testutil.FakeRequestHeaders method), 36
- GPULookupError, 29
- graphics_cards() (in module pyfarm.agent.sysinfo.graphics), 29
- ## H
- handle_stderr_line() (pyfarm.jobtypes.core.jobtype.JobType method), 48
- handle_stdout_line() (pyfarm.jobtypes.core.jobtype.JobType method), 48
- HIDDEN_FIELDS (pyfarm.agent.http.system.Configuration attribute), 28
- hostname() (in module pyfarm.agent.sysinfo.network), 30
- HTTP_REQUEST_SUCCESS (pyfarm.agent.testutil.BaseRequestTestCase attribute), 37
- http_retry_delay() (in module pyfarm.agent.http.core.client), 24
- HTTP_SCHEME (pyfarm.agent.testutil.BaseRequestTestCase attribute), 37
- HTTPLog (class in pyfarm.agent.http.core.client), 24
- ## I
- idle_time() (in module pyfarm.agent.sysinfo.cpu), 29
- Index (class in pyfarm.agent.http.system), 28
- InMemoryCache (class in pyfarm.agent.http.core.template), 27
- instance_class() (pyfarm.agent.testutil.BaseHTTPTestCase method), 37
- InsufficientSpaceError, 41
- interfaces() (in module pyfarm.agent.sysinfo.network), 30
- interpreter_architecture() (in module pyfarm.agent.sysinfo.system), 31

`interrupt()` (pyfarm.jobtypes.core.process.ProcessProtocol method), 51

`iowait()` (in module pyfarm.agent.sysinfo.cpu), 29

`ip()` (in module pyfarm.agent.entrypoints.parser), 19

`is_administrator()` (in module pyfarm.agent.sysinfo.user), 31

`is_successful()` (pyfarm.jobtypes.core.jobtype.JobType method), 47

`isLeaf` (pyfarm.agent.http.api.assign.Assign attribute), 21

`isLeaf` (pyfarm.agent.http.api.base.APIResource attribute), 21

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

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

`isLeaf` (pyfarm.agent.http.api.config.Config attribute), 22

`isLeaf` (pyfarm.agent.http.api.state.Restart attribute), 22

`isLeaf` (pyfarm.agent.http.api.state.Status attribute), 22

`isLeaf` (pyfarm.agent.http.api.state.Stop attribute), 22

`isLeaf` (pyfarm.agent.http.api.update.Update attribute), 23

J

`JobType` (class in pyfarm.jobtypes.core.jobtype), 43

`JOBTYPED_VERSION_URL` (pyfarm.jobtypes.core.internals.Cache attribute), 41

`json()` (pyfarm.agent.http.core.client.Response method), 25

`json_safe()` (in module pyfarm.agent.utility), 38

K

`kill()` (pyfarm.jobtypes.core.process.ProcessProtocol method), 51

L

`lineReceived()` (pyfarm.agent.manhole.LoggingManhole method), 33

`load()` (in module pyfarm.agent.sysinfo.cpu), 29

`load()` (pyfarm.agent.http.core.template.Loader class method), 28

`load()` (pyfarm.agent.utility.AgentUUID class method), 39

`load()` (pyfarm.jobtypes.core.jobtype.JobType class method), 43

`load_bytecode()` (pyfarm.agent.http.core.template.InMemoryCache method), 27

`LOAD_DATA_FOR_METHODS` (pyfarm.agent.http.core.resource.Resource attribute), 26

`Loader` (class in pyfarm.agent.http.core.template), 28

`log` (pyfarm.agent.utility.AgentUUID attribute), 39

`log_identifier` (pyfarm.jobtypes.core.internals.ProcessData attribute), 41

`log_stderr_line()` (pyfarm.jobtypes.core.jobtype.JobType method), 50

`log_stdout_line()` (pyfarm.jobtypes.core.jobtype.JobType method), 50

`logging` (pyfarm.jobtypes.core.internals.Process attribute), 42

`LoggingConfiguration` (class in pyfarm.agent.config), 31

`LoggingManhole` (class in pyfarm.agent.manhole), 33

`longMessage` (pyfarm.agent.testutil.TestCase attribute), 36

M

`mac_addresses()` (in module pyfarm.agent.sysinfo.network), 30

`machine_architecture()` (in module pyfarm.agent.sysinfo.system), 31

`manhole_factory()` (in module pyfarm.agent.manhole), 34

`map_path()` (pyfarm.jobtypes.core.jobtype.JobType method), 46

`master_contacted()` (pyfarm.agent.config.LoggingConfiguration method), 32

`mb()` (in module pyfarm.agent.http.system), 28

`methods` (pyfarm.agent.http.core.resource.Resource attribute), 26

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

`MODIFIED` (pyfarm.agent.config.LoggingConfiguration attribute), 32

N

`NAMESPACE` (pyfarm.agent.manhole.TelnetRealm attribute), 33

`next()` (pyfarm.agent.utility.UnicodeCSVReader method), 38

`next()` (pyfarm.agent.utility.UTF8Recoder method), 38

`node()` (pyfarm.jobtypes.core.jobtype.JobType method), 44

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

O

`operating_system()` (in module pyfarm.agent.sysinfo.system), 31

`outReceived()` (pyfarm.jobtypes.core.process.ProcessProtocol method), 51

P

`PERSISTENT_JOB_DATA` (pyfarm.jobtypes.core.jobtype.JobType attribute), 43

`pid` (pyfarm.jobtypes.core.process.ProcessProtocol attribute), 51

`pop()` (pyfarm.agent.config.LoggingConfiguration method), 32

`POP_CONFIG_KEYS` (pyfarm.agent.testutil.TestCase attribute), 36

- port() (in module pyfarm.agent.entrpoints.parser), 20
- post() (pyfarm.agent.http.api.assign.Assign method), 21
- post() (pyfarm.agent.http.api.state.Restart method), 22
- post() (pyfarm.agent.http.api.state.Stop method), 22
- post() (pyfarm.agent.http.api.update.Update method), 23
- post_agent_to_master() (pyfarm.agent.service.Agent method), 35
- post_cpu_count() (pyfarm.agent.service.Agent method), 35
- post_free_ram() (pyfarm.agent.service.Agent method), 35
- post_shutdown_to_master() (pyfarm.agent.service.Agent method), 35
- prepare_config() (pyfarm.agent.testutil.TestCase method), 36
- prepare_for_job() (pyfarm.jobtypes.core.jobtype.JobType class method), 43
- preprocess_stderr_line() (pyfarm.jobtypes.core.jobtype.JobType method), 49
- preprocess_stdout_line() (pyfarm.jobtypes.core.jobtype.JobType method), 49
- Process (class in pyfarm.jobtypes.core.internals), 41
- process (pyfarm.jobtypes.core.process.ProcessProtocol attribute), 51
- process_memory() (in module pyfarm.agent.sysinfo.memory), 30
- process_output() (pyfarm.jobtypes.core.jobtype.JobType method), 47
- PROCESS_PROTOCOL (pyfarm.jobtypes.core.jobtype.JobType attribute), 43
- process_started() (pyfarm.jobtypes.core.jobtype.JobType method), 47
- process_stderr_line() (pyfarm.jobtypes.core.jobtype.JobType method), 50
- process_stdout_line() (pyfarm.jobtypes.core.jobtype.JobType method), 50
- process_stopped() (pyfarm.jobtypes.core.jobtype.JobType method), 47
- ProcessData (class in pyfarm.jobtypes.core.internals), 41
- processEnded() (pyfarm.jobtypes.core.process.ProcessProtocol method), 51
- ProcessProtocol (class in pyfarm.jobtypes.core.process), 51
- protocol (pyfarm.jobtypes.core.internals.ProcessData attribute), 41
- psutil_process (pyfarm.jobtypes.core.process.ProcessProtocol attribute), 51
- putChild() (pyfarm.agent.http.core.resource.Resource method), 26
- pyfarm.agent (module), 39
- pyfarm.agent.config (module), 31
- pyfarm.agent.entrpoints (module), 21
- pyfarm.agent.entrpoints.development (module), 19
- pyfarm.agent.entrpoints.main (module), 19
- pyfarm.agent.entrpoints.parser (module), 19
- pyfarm.agent.entrpoints.supervisor (module), 21
- pyfarm.agent.entrpoints.utility (module), 21
- pyfarm.agent.http (module), 29
- pyfarm.agent.http.api (module), 23
- pyfarm.agent.http.api.assign (module), 21
- pyfarm.agent.http.api.base (module), 21
- pyfarm.agent.http.api.config (module), 22
- pyfarm.agent.http.api.state (module), 22
- pyfarm.agent.http.api.tasklogs (module), 23
- pyfarm.agent.http.api.tasks (module), 23
- pyfarm.agent.http.api.update (module), 23
- pyfarm.agent.http.core (module), 28
- pyfarm.agent.http.core.client (module), 24
- pyfarm.agent.http.core.resource (module), 26
- pyfarm.agent.http.core.server (module), 26
- pyfarm.agent.http.core.template (module), 27
- pyfarm.agent.http.system (module), 28
- pyfarm.agent.manhole (module), 33
- pyfarm.agent.service (module), 34
- pyfarm.agent.sysinfo (module), 31
- pyfarm.agent.sysinfo.cpu (module), 29
- pyfarm.agent.sysinfo.graphics (module), 29
- pyfarm.agent.sysinfo.memory (module), 30
- pyfarm.agent.sysinfo.network (module), 30
- pyfarm.agent.sysinfo.system (module), 31
- pyfarm.agent.sysinfo.user (module), 31
- pyfarm.agent.testutil (module), 36
- pyfarm.agent.utility (module), 38
- pyfarm.jobtypes (module), 52
- pyfarm.jobtypes.core (module), 52
- pyfarm.jobtypes.core.internals (module), 41
- pyfarm.jobtypes.core.jobtype (module), 42
- pyfarm.jobtypes.core.process (module), 51
- pyfarm.jobtypes.examples (module), 52
- PythonHelloWorld (class in pyfarm.jobtypes.examples), 52
- Q
- quote() (pyfarm.agent.http.core.client.HTTPLog static method), 24
- quote_url() (in module pyfarm.agent.utility), 38
- R
- RAND_LENGTH (pyfarm.agent.testutil.TestCase attribute), 36
- random() (in module pyfarm.agent.entrpoints.development), 19
- random() (in module pyfarm.agent.http.core.client), 26

random_port() (in module pyfarm.agent.testutil), 36
reannounce() (pyfarm.agent.service.Agent method), 34
REDIRECT_TARGET (pyfarm.agent.testutil.BaseRequestTestCase attribute), 37
register_callback() (pyfarm.agent.config.ConfigurationWithCallbacks class method), 33
render() (pyfarm.agent.http.core.resource.Resource method), 26
render() (pyfarm.agent.http.core.server.StaticPath method), 27
render() (pyfarm.agent.http.core.template.DeferredTemplateset method), 28
repeating_call() (pyfarm.agent.service.Agent method), 34
REPLACE_REPEATED_DELIMITER (pyfarm.agent.http.core.server.RewriteRequest attribute), 27
ReplaceEnvironment (class in pyfarm.jobtypes.core.process), 51
Request (class in pyfarm.agent.http.core.client), 25
request() (in module pyfarm.agent.http.core.client), 25
request_from_master() (in module pyfarm.agent.utility), 38
requestAvatar() (pyfarm.agent.manhole.TelnetRealm method), 33
requestFactory (pyfarm.agent.http.core.server.Site attribute), 27
requestReceived() (pyfarm.agent.http.core.server.RewriteRequest method), 27
requires_master() (in module pyfarm.agent.testutil), 36
RESOLVED_DNS_NAME (pyfarm.agent.testutil.BaseRequestTestCase attribute), 37
Resource (class in pyfarm.agent.http.core.resource), 26
Response (class in pyfarm.agent.http.core.client), 25
response() (pyfarm.agent.http.core.client.HTTPLog static method), 24
response() (pyfarm.agent.testutil.FakeRequest method), 36
Restart (class in pyfarm.agent.http.api.state), 22
retry() (pyfarm.agent.http.core.client.Request method), 25
RewriteRequest (class in pyfarm.agent.http.core.server), 27
RFC
 RFC 1918, 30
running() (pyfarm.jobtypes.core.process.ProcessProtocol method), 51

S
save() (pyfarm.agent.utility.AgentUUID class method), 39
SCHEMAS (pyfarm.agent.http.api.assign.Assign attribute), 21
SCHEMAS (pyfarm.agent.http.api.state.Stop attribute), 22
SCHEMAS (pyfarm.agent.http.api.update.Update attribute), 23
SCHEMAS (pyfarm.agent.http.core.resource.Resource attribute), 26
seconds() (in module pyfarm.agent.http.system), 28
set_default_environment() (pyfarm.jobtypes.core.jobtype.CommandData method), 42
set_states() (pyfarm.jobtypes.core.jobtype.JobType method), 46
set_task_state() (pyfarm.jobtypes.core.jobtype.JobType method), 46
setResponseCode() (pyfarm.agent.testutil.FakeRequest method), 36
setUp() (pyfarm.agent.testutil.BaseAPITestCase method), 37
setUp() (pyfarm.agent.testutil.BaseHTMLTestCase method), 37
setUp() (pyfarm.agent.testutil.BaseHTTPTestCase method), 37
setUp() (pyfarm.agent.testutil.BaseRequestTestCase method), 37
setUp() (pyfarm.agent.testutil.TestCase method), 36
should_reannounce() (pyfarm.agent.service.Agent method), 34
show() (in module pyfarm.agent.manhole), 33
shutting_down (pyfarm.agent.service.Agent attribute), 34
sigint_handler() (pyfarm.agent.service.Agent method), 35
Site (class in pyfarm.agent.http.core.server), 27
skipIf (class in pyfarm.agent.testutil), 36
spawn_persistent_process() (pyfarm.jobtypes.core.jobtype.JobType class method), 44
start() (pyfarm.agent.entrypoints.main.AgentEntryPoint method), 19
start() (pyfarm.agent.service.Agent method), 35
start() (pyfarm.jobtypes.core.jobtype.JobType method), 46
start_daemon_posix() (in module pyfarm.agent.entrypoints.utility), 21
start_deferred (pyfarm.jobtypes.core.internals.Process attribute), 42
started (pyfarm.jobtypes.core.internals.ProcessData attribute), 41
StaticPath (class in pyfarm.agent.http.core.server), 27
Status (class in pyfarm.agent.http.api.state), 22
status() (pyfarm.agent.entrypoints.main.AgentEntryPoint method), 19
Stop (class in pyfarm.agent.http.api.state), 22

- stop() (pyfarm.agent.entrypoints.main.AgentEntryPoint method), 19
- stop() (pyfarm.agent.service.Agent method), 35
- stop() (pyfarm.agent.testutil.FakeAgent method), 36
- stop() (pyfarm.jobtypes.core.jobtype.JobType method), 46
- stopped (pyfarm.jobtypes.core.internals.ProcessData attribute), 41
- stopped_deferred (pyfarm.jobtypes.core.internals.Process attribute), 42
- StoreAction (in module pyfarm.agent.entrypoints.parser), 20
- StoreConstAction (in module pyfarm.agent.entrypoints.parser), 20
- StoreFalseAction (in module pyfarm.agent.entrypoints.parser), 20
- StoreTrueAction (in module pyfarm.agent.entrypoints.parser), 20
- SubParsersAction (in module pyfarm.agent.entrypoints.parser), 20
- supervisor() (in module pyfarm.agent.entrypoints.supervisor), 21
- System (class in pyfarm.jobtypes.core.internals), 42
- system_data() (pyfarm.agent.service.Agent method), 34
- system_time() (in module pyfarm.agent.sysinfo.cpu), 29
- ## T
- TaskLogs (class in pyfarm.agent.http.api.tasklogs), 23
- Tasks (class in pyfarm.agent.http.api.tasks), 23
- TASKS_SCHEMA() (in module pyfarm.agent.utility), 38
- TelnetRealm (class in pyfarm.agent.manhole), 33
- tempdir() (pyfarm.jobtypes.core.jobtype.JobType method), 45
- TEMPLATE (pyfarm.agent.http.core.resource.Resource attribute), 26
- template (pyfarm.agent.http.core.resource.Resource attribute), 26
- TEMPLATE (pyfarm.agent.http.system.Configuration attribute), 28
- TEMPLATE (pyfarm.agent.http.system.Index attribute), 28
- template_class (pyfarm.agent.http.core.template.Environment attribute), 28
- terminate() (pyfarm.jobtypes.core.process.ProcessProtocol method), 51
- test_content_types() (pyfarm.agent.testutil.BaseHTTPTestCase method), 37
- test_implements_methods() (pyfarm.agent.testutil.BaseHTTPTestCase method), 37
- test_instance() (pyfarm.agent.testutil.BaseHTTPTestCase method), 37
- test_leaf() (pyfarm.agent.testutil.BaseHTTPTestCase method), 37
- test_methods_exist_for_schema() (pyfarm.agent.testutil.BaseHTTPTestCase method), 37
- test_missing_schemas() (pyfarm.agent.testutil.BaseHTTPTestCase method), 37
- test_parent() (pyfarm.agent.testutil.BaseAPITestCase method), 37
- test_template_loaded() (pyfarm.agent.testutil.BaseHTMLTestCase method), 37
- test_template_set() (pyfarm.agent.testutil.BaseHTMLTestCase method), 37
- TEST_URL (pyfarm.agent.testutil.BaseRequestTestCase attribute), 37
- TestCase (class in pyfarm.agent.testutil), 36
- timeout (pyfarm.agent.testutil.TestCase attribute), 36
- total_consumption() (in module pyfarm.agent.sysinfo.memory), 30
- total_cpus() (in module pyfarm.agent.sysinfo.cpu), 29
- total_ram() (in module pyfarm.agent.sysinfo.memory), 30
- total_seconds() (in module pyfarm.agent.utility), 38
- TransportProtocolFactory (class in pyfarm.agent.manhole), 33
- TYPE_MAPPING (pyfarm.agent.entrypoints.parser.ActionMixin attribute), 20
- TypeChecks (class in pyfarm.jobtypes.core.internals), 42
- ## U
- uidgid() (in module pyfarm.agent.entrypoints.parser), 20
- UnicodeCSVReader (class in pyfarm.agent.utility), 38
- UnicodeCSVWriter (class in pyfarm.agent.utility), 38
- Update (class in pyfarm.agent.http.api.update), 23
- update() (pyfarm.agent.config.LoggingConfiguration method), 32
- uptime() (in module pyfarm.agent.sysinfo.system), 31
- URI (pyfarm.agent.testutil.BaseHTTPTestCase attribute), 37
- used_ram() (in module pyfarm.agent.sysinfo.memory), 30
- user_time() (in module pyfarm.agent.sysinfo.cpu), 29
- username() (in module pyfarm.agent.sysinfo.user), 31
- UTF8Recoder (class in pyfarm.agent.utility), 38
- uuid (pyfarm.jobtypes.core.process.ProcessProtocol attribute), 51
- uuid_type() (in module pyfarm.agent.entrypoints.parser), 20

V

`validate()` (pyfarm.jobtypes.core.jobtype.CommandData method), [42](#)
`validate_environment()` (in module pyfarm.agent.utility), [38](#)
`validate_uuid()` (in module pyfarm.agent.utility), [38](#)
`Versions` (class in pyfarm.agent.http.api.base), [22](#)

W

`write()` (pyfarm.agent.http.core.server.RewriteRequest method), [27](#)
`write()` (pyfarm.agent.testutil.FakeRequest method), [36](#)
`writerow()` (pyfarm.agent.utility.UnicodeCSVWriter method), [38](#)
`writerows()` (pyfarm.agent.utility.UnicodeCSVWriter method), [38](#)