pyfarm.agent Documentation

Release 0.8.3

Oliver Palmer, Guido Winkelmann

March 12, 2015

Contents

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

CHAPTER 1

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
                        shoulduse [default: None]

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

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

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

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

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

usage: pyfarm-agent [status|start|stop] start [-h] [--state STATE]
                                              [--time-offset TIME_OFFSET]
                                              [--ntp-server NTP_SERVER]
                                              [--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-offset HTTP_RETRY_DELAY_OFFSET]
                                                [--http-retry-delay-factor HTTP_RETRY_DELAY_FACTOR]
                                                [--jobtype-no-cache]

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

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

  --state STATE         The current agent state, valid values are ['disabled',
                        'offline', 'running', 'online']. [default: online]
  --time-offset TIME_OFFSET
                        If provided then don't talk to the NTP server at all
                        to calculate the time offset. If you know for a fact
                        that this host's time is always up to date then
                        setting this to 0 is probably a safe bet.
  --ntp-server NTP_SERVER
                        The default network time server this agent should
                        query to retrieve the real time. This will be used to
                        help determine the agent's clock skew if any. Setting
                        this value to '' will effectively disable this query.
                        [default: None]
  --ntp-server-version NTP_SERVER_VERSION
                        The version of the NTP server in case it's running an
                        olderor 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-offset HTTP_RETRY_DELAY_OFFSET
                        If a http request to the master has failed, wait at
                        least this amount of time before resending the
                        request.
  --http-retry-delay-factor HTTP_RETRY_DELAY_FACTOR
                        The value provided here is used in combination with
                        --http-retry-delay-offset to calculate the retry
                        delay. This is used as a multiplier against random()
                        before being added to the offset.

Job Types:
  --jobtype-no-cache    If provided then do not cache job types, always
                        directly retrieve them. This is beneficial if you're
                        testing the agent or a new job type class.

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

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

optional flags:
  Flags that control how the agent is stopped

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

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

```
Start and monitor the agent process

optional arguments:
  -h, --help            show this help message and exit
  --updates-drop-dir UPDATES_DROP_DIR
                        Where to look for agent updates
  --agent-package-dir AGENT_PACKAGE_DIR
                        Path to the actual agent code
  --pidfile PIDFILE     The file to store the process id in. [default: None]
  -n, --no-daemon       If provided then do not run the process in the
                        background.
  --chdir CHDIR         The directory to chdir to upon launch.
  --uid UID             The user id to run the supervisor as. *This setting is
                        ignored on Windows.*
  --gid GID             The group id to run the supervisor as. *This setting
                        is ignored on Windows.*
```

1.2 Development Commands

1.2.1 pyfarm-dev-fakerender

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

Very basic command line tool which vaguely simulates a render.

optional arguments:
  -h, --help            show this help message and exit
  --ram RAM             How much ram in megabytes the fake command should
                        consume
  --duration DURATION   How many seconds it should take to run this command
  --return-code RETURN_CODE
                        The return code to return, declaring this flag
                        multiple times will result in a random return code.
                        [default: [0]]
  --duration-jitter DURATION_JITTER
                        Randomly add or subtract this amount to the total
                        duration
  --ram-jitter RAM_JITTER
                        Randomly add or subtract this amount to the ram
  -s START, --start START
                        The start frame. If no other flags are provided this
                        will also be the end frame.
  -e END, --end END     The end frame
  -b BY, --by BY        The by frame
  --spew                Spews lots of random output to stdout which is
                        generally a decent stress test for log processing
                        issues. Do note however that this will disable the
                        code which is consuming extra CPU cycles. Also, use
                        this option with care as it can generate several
                        gigabytes of data per frame.
```

```
  --segfault            If provided then there's a 25% chance of causing a
                        segmentation fault.
```

1.2.2 pyfarm-dev-fakework

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

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

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

CHAPTER 2

Environment Variables

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

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

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

CHAPTER 3

Configuration Files

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

3.1 Agent

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

```
# The platform specific locations where the agent uuid
# file is default.  This can be overridden with --agent-id-file flag.
agent_id_file_platform_defaults:
  linux: /etc/pyfarm/agent/uuid.dat
  mac: /Library/pyfarm/agent/uuid.dat
  bsd: /etc/pyfarm/agent/uuid.dat
  windows: $LOCALAPPDATA/pyfarm/agent/uuid.dat

# The platform specific locations where the run control file is looked for
run_control_file_by_platform:
  linux: /tmp/pyfarm/agent/should_be_running
  mac: /tmp/pyfarm/agent/should_be_running
  bsd: /tmp/pyfarm/agent/should_be_running
  windows: $TEMP/pyfarm/agent/should_be_running

# The default location to store data.  $temp will expand to
# whatever pyfarm's data root is plus the application
# name (agent).  For example on Linux this would expand to
# /tmp/pyfarm/agent
agent_data_root: $temp

# Defines the number of seconds between iterations of pyfarm-supervisor's
# agent status check.
supervisor_interval: 5

# The location where the agent should change directories
# into upon starting.  If this value is not set then no
# changes will be made.
agent_chdir:

# The location where static web files should be served from.  This
```

```
# will default to using PyFarm's installation root.
agent_static_root: auto

# The default location where lock files should be stored.  By
# default these will be stored alone side other data
# inside the `agent_data_root` value above.
lock_file_root: $agent_data_root/lock

# Locations of specific lock files
agent_lock_file: $lock_file_root/agent.pid
supervisor_lock_file: $lock_file_root/supervisor.pid

# Where user data for the agent is stored.  ~ will be expanded
# to the current users's home directory.
agent_user_data: ~/.pyfarm/agent

# The default location where the agent should save logs to.  This
# includes both logs from processes and the agent log itself.
agent_logs_root: $agent_data_root/logs

# The location where agent updates should be stored.
agent_updates_dir: $agent_data_root/updates

# The default port which the agent should use to serve the
# REST api.
agent_api_port: 50000

# The location where the the agent should save its own
# logging output to.
agent_log: $agent_logs_root/agent.log
supervisor_log: $agent_logs_root/supervisor.log

# The user agent the master will use when connecting to the agent's
# REST api.  This value should only be changed if the master's code
# is updated with a new user agent.  Change this value has not effect
# on the master.
master_user_agent: PyFarm/1.0 (master)

# Configuration values which control how the url
# for the master is constructed.  If 'master' is not set
# the --master flag will be required to start the agent.
master:
master_api_version: 1
master_api: http://$master/api/v$master_api_version

# The user agent the master uses to talke to the agent's
# REST api.  This value should not be modified unless
# there's a specific reason to do so.
master_user_agent: PyFarm/1.0 (master)

# 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.
agent_master_reannounce: 120

# How many seconds the agent should spend attempting to inform
```

```
# the master that it's shutting down.
agent_shutdown_timeout: 15

# Number of seconds to offset from zero when calculating the
# http retry delay.
agent_http_retry_delay_offset: 1

# If an http request fails, use this as the base value
# to help determine how long we should wait before retrying.  This
# value is then used to calculate the final delay:
#   agent_http_retry_delay_offset + (random() * agent_http_retry_delay_factor)
agent_http_retry_delay_factor: 5

# Controls if the http client connection should be persistent or
# not.  Generally this should always be True because the connection
# self-terminates after a short period of time anyway.  For higher
# latency situations or with larger deployments this value should
# be False.
agent_http_persistent_connections: True

# If True then html templates will be reloaded with
# every request instead of cached.
agent_html_template_reload: False

# If True then reformat json output to be more human
# readable.
agent_pretty_json: True

# How often the agent should check for changes in ram.  This value
# is used to ensure ram usage is checked at least this often though
# it may be checked more often due to other events (such as jobs
# running)
agent_ram_check_interval: 30

# If the ram has changed this may megabytes since the last
# check then report the change to the master.
agent_ram_report_delta: 100

# How much the agent should wait, in seconds, between
# each report about a change in ram.
agent_ram_max_report_frequency: 10

# The default network time server and version the agent
# should use to calcuate its clock skew.
agent_ntp_server: pool.ntp.org
agent_ntp_server_version: 2

# The amount of time this agent is offset from what
# would be considered correct based on an atomic
# clock.  If this value is set to auto the time will
# be calculated using NTP.
agent_time_offset: auto

# Physical and network information about the host the agent
# is running on.  Setting these values to 'auto' will cause
# them to be initilized to the system's current
# configuration values.
agent_ram: auto
```

```
agent_cpus: auto
agent_hostname: auto

# When True this will enable a telnet connection
# to the agent which will present a Python interpreter
# upon connection.  This is mainly used for debugging
# and direct manipulation of the agent.  You can use
# the show() function once connected to see what
# objects are available.
agent_manhole: False
agent_manhole_port: 50001
agent_manhole_username: admin
agent_manhole_password: admin

# NOTE: The following values are used by the unittests and should be
# generally ignored for anything other than development.
agent_unittest:
  dns_test_hostname: example.com
  client_redirect_target: http://example.com
  client_api_test_url_https: https://httpbin.pyfarm.net
  client_api_test_url_http: http://httpbin.pyfarm.net

# A list of paths or names where the 'lspci' command can
# be called from on Linux.  This is used to retrieve information
# about graphics cards installed on the system in
# 'pyfarm.agent.sysinfo.graphics.graphics_cards'.
# If you need run the command with sudo you may also specify an entry
# like this:
#  - sudo lspci
sysinfo_command_lspci:
  - lspci
  - /bin/lspci
  - /sbin/lspci
  - /usr/sbin/lspci
  - /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.

```
# When set to True caching of job types will be enabled.  When set to
# False caching is disabled and every job type will retrieved from
# the master directly.
jobtype_enable_cache: True

# If True then output from all processes will be sent directly to
# the agent's logger(s) instead of to the log file assoicated
# with each process.
jobtype_capture_process_output: False

# The location where tasks should be logged
jobtype_task_logs: $agent_logs_root/tasks

# The filename to an individual log file.  This filename supports several
# internal variables:
```

```
#
#   $YEAR - The current year
#   $MONTH - The current month
#   $DAY - The current day
#   $HOUR - The current hour
#   $MINUTE - The current hour
#   $JOB - The id of the job this log is for
#   $PROCESS - The uuid of the process object responsible for creating the log
#
# In addition to the above you can, as with any configuration variable,
# also use environment variables in the filename.
# Path separators ("/" and "\") are not allowed.
jobtype_task_log_filename:
    $YEAR-$MONTH-$DAY_$HOUR-$MINUTE-$SECOND_$JOB_$PROCESS.csv

# store cached source code from the master.  Note
# that $temp will be expanded to the local system's
# temp directory.  If this directory does not exist
# it will be created.  Leaving this value blank will
# disable job type caching.
jobtype_cache_directory: $temp/jobtype_cache

# The root directory that the default implementation of JobType.tempdir()
# will create a path using tempfile.mkdtemp.
jobtype_tempdir_root: $temp/tempdir/$JOBTYPE_UUID

# If True then expand environment variables in file paths.
jobtype_expandvars: True

# If True, then ignore any errors produced when tring
# to map users and groups to IDs.  This will cause the
# underlying methods in the job type to instead run
# as the job type's owner instead, ignoring what the
# incoming job requests.
# NOTE: This value is not used on Windows.
jobtype_ignore_id_mapping_errors: False

# Any additional key/value pairs to include
# in the environment of a process launched
# by a job type.
jobtype_default_environment: {}


# Configures the thread pool used by job types
# for logging.
jobtype_logging_threadpool:
  # Setting this value to something smaller than `1` will result
  # in an exception being raised.  This value also cannot be larger
  # than `max_threads` below.
  min_threads: 3

  # This value must be greater than or equal to `min_threads`
  # above.  You may also set this value to 'auto' meaning the
  # number of processors times 1.5 or 20 (whichever is lower).
  max_threads: auto

  # As log messages are sent from processes they are stored
  # 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
```

CHAPTER 4

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`**`()` → 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 0x7f1b80b400c8>: <function direxists at 0x7f1b794bdc08>, <function isfile at 0x7f1

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 0x7f1b78e7bf90>}

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 0x7f1b78a680d0>}

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 0x7f1b78965190>}

isLeaf = False

post(***kwargs*)

Module contents

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

pyfarm.agent.http.core package

Submodules

pyfarm.agent.http.core.client module

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

class pyfarm.agent.http.core.client.**HTTPLog**
Bases: object

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

static queue(*method*, *url*, *uid=None*)
Logs the request we're asking treq to queue

static response(*response*, *uid=None*)
Logs the return code of a request that treq completed

static error(*failure*, *uid=None*, *method=None*, *url=None*)
Called when the treq request experiences an error and calls the errback method.

pyfarm.agent.http.core.client.**build_url**(*url*, *params=None*)
Builds the full url when provided the base url and some url parameters:

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

Parameters

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

pyfarm.agent.http.core.client.**http_retry_delay**(*offset=None*, *factor=None*, *rand=None*)
Returns a floating point value that can be used to delay an http request. The main purpose of this is to ensure that not all requests are run with the same interval between them.

The basic formula for the retry delay is:

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

Parameters

- **factor** (*int or float*) – The factor to multiply the output from random() by.

  This defaults to the agent_http_retry_delay_factor configuration variable.
- **offset** – The initial offset to start the calculation at.

  This defaults to the agent_http_retry_delay_offset configuration variable.
- **rand** – A callable to determine randomness, defaulting to random(). This is mainly used for testing purposes.

class pyfarm.agent.http.core.client.**Request**
Bases: pyfarm.agent.http.core.client.Request

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

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

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

class pyfarm.agent.http.core.client.**Response**(*deferred*, *response*, *request*)
Bases: twisted.internet.protocol.Protocol

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

Parameters

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

data()
Returns the data currently contained in the buffer.

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

json(*loader=<function loads at 0x7f1b7d475488>*)
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 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 0x7f1b7890c390>

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', 'master_api', 'master', 'agent_ram_check_interval', 'agent_ram

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 0x7f1b80ace480>*, *old_value=<object object at 0x7f1b80ace480>*)

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 0x7f1b80ace480>*, *old_value=<object object at 0x7f1b80ace480>*)
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 instanced 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 0x7f1b80ace480>*)
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 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 may 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(*force=False*)
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(**args*, ***kwargs*)
Internal code which stops the agent. This will terminate any running processes, inform the master of the terminated tasks, update the state of the agent on the master.

sigint_handler(*_)

post_shutdown_to_master(**args*, ***kwargs*)
This method is called before the reactor shuts down and lets the master know that the agent's state is now `offline`

post_agent_to_master(**args*, ***kwargs*)
Runs the POST request to contact the master. Running this method multiple times should be considered safe but is generally something that should be avoided.

callback_agent_id_set(*change_type*, *key*, *new_value*, *old_value*, *shutdown_events=True*)
When *agent_id* is created we need to:

•Register a shutdown event so that when the agent is told to shutdown it will notify the master of a state change.

•Star the scheduled task manager

callback_shutting_down_changed(*change_type*, *key*, *new_value*, *old_value*)
When *shutting_down* is changed in the configuration, set or reset self.shutdown_timeout

4.2.4 pyfarm.agent.testutil module

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

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

pyfarm.agent.testutil.**random_port**(*bind='127.0.0.1'*)
Returns a random port which is not in use

pyfarm.agent.testutil.**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._asynctest.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()

4.2.5 pyfarm.agent.utility module

Utilities

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

pyfarm.agent.utility.**validate_environment**(*values*)
Ensures that `values` is a dictionary and that it only contains string keys and values.

pyfarm.agent.utility.**validate_uuid**(*value*)
Ensures that `value` can be converted to or is a UUID object.

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

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

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

pyfarm.agent.utility.**quote_url**(*source_url*)
This function serves as a wrapper around `urlsplit()` and `quote()` and a url that has the path quoted.

pyfarm.agent.utility.**dumps**(**args*, ***kwargs*)
Agent's implementation of `json.dumps()` or `pyfarm.master.utility.jsonify()`

pyfarm.agent.utility.**request_from_master**(*request*)
Returns True if the request appears to be coming from the master

class pyfarm.agent.utility.**UTF8Recoder**(*f*, *encoding*)
Bases: `object`

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

next()

class pyfarm.agent.utility.**UnicodeCSVReader**(*f*, *dialect=<class csv.excel at 0x7f1b7ab29390>*, *encoding='utf-8'*, ***kwds*)
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 0x7f1b7ab29390>*, ***kwds*)
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 0x7f1b7a904ed0>

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

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

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

pyfarm.agent.utility.**remove_file**(*path*, *retry_on_exit=False*, *raise_=True*, *ignored_errnos=(2*, *)*)
Simple function to remove the provided file or retry on exit if requested. This function standardizes the

log output, ensures it's only called once per path on exit and handles platform specific exceptions (ie. `WindowsError`).

Parameters

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

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

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

Parameters

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

4.3 Module contents

4.3.1 PyFarm Agent

Core module containing code to run PyFarm's agent.

CHAPTER 5

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)

`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 modle.

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 alosed 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 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 instanced. 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 0x7f1b78e7bbd0>

classmethod load(*assignment*)

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

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

classmethod prepare_for_job(*job*)

Note: This method is not yet implemented

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

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

By default this method does nothing.

classmethod **cleanup_after_job**(*persistent_data*)

Note: This method is not yet implemented

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

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

classmethod **spawn_persistent_process**(*job*, *command_data*)

Note: This method is not yet implemented

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

node()

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

- **master_api** - The base url the agent is using to communicate with the master.
- **hostname** - The hostname as reported to the master.
- **agent_id** - The unique identifier used to identify. this agent to the master.
- **id** - The database id of the agent as given to us by the master on startup of the agent.
- **cpus** - The number of CPUs reported to the master
- **ram** - The amount of ram reported to the master.
- **total_ram** - The amount of ram, in megabytes, that's installed on the system regardless of what was reported to the master.
- **free_ram** - How much ram, in megabytes, is free for the entire system.
- **consumed_ram** - How much ram, in megabytes, is being consumed by the agent and any processes it has launched.
- **admin** - Set to True if the current user is an administrator or 'root'.
- **user** - The username of the current user.
- **case_sensitive_files** - True if the file system is case sensitive.
- **case_sensitive_env** - True if environment variables are case sensitive.
- **machine_architecture** - The architecture of the machine the agent is running on. This will return 32 or 64.

•**operating_system** - The operating system the agent is executing on. This value will be 'linux', 'mac' or 'windows'. In rare circumstances this could also be 'other'.

Raises KeyError

Raised if one or more keys are not present in the global configuration object.

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

assignments()
Short cut method to access tasks

tempdir(*new=False*, *remove_on_finish=True*)
Returns a temporary directory to be used within a job type. By default once called the directory will be created on disk and returned from this method.

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

Parameters

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

get_uid_gid(*user*, *group*)
Overridable. This method to convert a named user and group into their respective user and group ids.

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

get_command_list(*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 of there was at least one failed process.

process_started(*protocol*)
Overridable. This method is called when a child process started running.

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

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

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

Parameters

- **protocol** (`ProcessProtocol`) – The protocol instance which produced `output`
- **output** (*string*) – The blob of text or line produced

- **line_fragments** (*dict*) – The line fragment dictionary containing individual line fragments. This will be either `self._stdout_line_fragments` or `self._stderr_line_fragments`.
- **line_handler** (*callable*) – The function to handle any lines produced. This will be either `handle_stdout_line()` or `handle_stderr_line()`

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

handle_stdout_line(*protocol*, *stdout*)

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

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

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

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

Parameters

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

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

handle_stderr_line(*protocol*, *stderr*)

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

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

- `preprocess_stderr_line()`
- `format_stderr_line()`
- `log_stderr_line()`
- `process_stderr_line()`

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

Parameters

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

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

preprocess_stdout_line(*protocol*, *stdout*)

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

The default implementation does nothing.

Parameters

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

Return type string

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

preprocess_stderr_line(*protocol*, *stderr*)

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

The default implementation does nothing.

Parameters

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

Return type string

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

format_stdout_line(*protocol*, *stdout*)

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

The default implementation does nothing.

Parameters

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

Return type string

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

format_stderr_line(*protocol*, *stderr*)

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

The default implementation does nothing.

Parameters

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

Return type string

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

log_stdout_line(*protocol*, *stdout*)

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

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

Parameters

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

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

log_stderr_line(*protocol*, *stderr*)

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

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

Parameters

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

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

process_stderr_line(*protocol*, *stderr*)

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

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

Parameters

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

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

process_stdout_line(*protocol*, *stdout*)

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

The default implementation does nothing.

Parameters

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

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

pyfarm.jobtypes.core.process module

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

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

Bases: object

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

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

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

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

Bases: twisted.internet.protocol.ProcessProtocol

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

uuid

pid

process
The underlying Twisted process object

psutil_process
Returns a psutil.Process object for the running process

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

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

outReceived(*data*)
Called when the process emits on stdout

errReceived(*data*)
Called when the process emits on stderr

kill()
Kills the underlying process, if running.

terminate()
Terminates the underlying process, if running.

interrupt()
Interrupts the underlying process, if running.

running()
Method to determine whether the child process is currently running

Module contents

5.2 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.

CHAPTER 6

Indices and tables

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

/api

p

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W