Logging

Types of Logs

Tractor generates several types of log files:

- **Engine Logs**

  The engine emits logs about its status and ongoing activities, such as new jobs being spooled, or commands being assigned to blades. These logs are written to the file named on the engine command-line with the tractor-engine --log=filenameoption.
- **Blade Logs**

  Each blade running on a server host emits logs about the status of the tractor-blade process itself and its ongoing activities, such as preparations to launch a command from a user job. These blade logs are written to the file named on the blade command-line with the tractor-blade --log=filename option.
- **Command Output Logs**

  Every command launched by a tractor-blade, such as prman, can also generate log output that Tractor captures. This includes anything that the launched program writes to its stdout or stderr. These are *command output logs* and they are treated specially by Tractor. See the notes on Command Output Logs below for more details.

Note: when the engine or blade is installed as a systemd service, logging is handled by journald by default and the --log option is not necessary.

Log Rotation

Engine logs: can be rotated using an external mechanism such as the Linux logrotate facility. The engine responds to SIGHUP by closing the file named by its --log command-line option, and then reopening that same filename to continue logging. The Linux logrotate facility, or similar, can rotate logs based on site-specified criteria by mv-ing the log file (renaming it, while Tractor keeps the same i-node open), then sending a SIGHUP to Tractor, causing a fresh log file to be created. NOTE: System configurations using logrotate on the engine logs, and which are also using the "compress" option, should also add the "delaycompress" option for the engine log handler. There is a race condition in logrotate when creating the gz file which can remove (rather than just rename) the old log before all of the engine's outstanding log transactions to the old log have completed. Here is an example of a configuration, e.g. /etc /logrotate.d/tractor-engine, assuming that the tractor-engine process itself has been started with the option *--log=/var/log/tractor-engine.log*:

```
/var/log/tractor-engine.log {
   size 25M
   rotate 5
   missingok
   postrotate
      /usr/bin/killall -HUP tractor-engine
   endscript
   compress
   compressoptions --best
   delaycompress
}
```

Blade logs: the diagnostic output of the tractor-blade.py itself are directed to the file specified with the --log option, or to stdout by default. When using files, they are automatically rotated when they reach approximately 25 megabytes in size. Rotation is performed by Python's logging.handlers. RotatingFileHandler. The tractor-blade command line option **--logrotate=NNN** sets the rotation threshold to NNN megabytes. This is usually the simplest approach.

In **rare** cases where advanced behavior is required, the blade's logging configuration can alternatively be specified using Python's built-in "logging.configs. fileConfig" scheme. The tractor-blade.py option --logconfig=FILE specifies an external configuration, which is assumed to be in format expected by Python's built-in logging module. Here is an example logging config file that implements the default logging scheme, although redirected to a specific file:

```
[loggers]
keys=root,tractorblade

[handlers]
keys=hroot,bhrotater

[formatters]
keys=tbfmt

[logger_root]
level=NOTSET
handlers=hroot

[logger_tractorblade]
level=INFO
handlers=bhrotater
propagate=0
qualname=tractor-blade

[formatter_tbfmt]
format=%(asctime)s %(levelname)-7s %(message)s
datefmt=%m/%d %H:%M:%S

[handler_hroot]
class=StreamHandler
level=NOTSET
formatter=tbfmt
args=(sys.stdout,)

[handler_bhrotater]
class=handlers.RotatingFileHandler
level=INFO
formatter=tbfmt
args=('/tmp/b.txt', 'a', 26214400, 5)
```

The output of applications **launched** by the blade, such as renders, are not rotated. There is one new log file created per task for these commands.

Command Output Logs: Details

How command output is saved

The output from individual commands launched by each tractor-blade is written to an per-task logfile. The blades will also filter the command output stream to extract *progress indicators* that are used to generate user-interface updates, such as percent-done bars.

The blade.config profile setting *CmdOutputLogging* controls how each type of blade performs those writes. Output logs can be delivered through a Tractor-specific TCP socket connection to a central Tractor (Python) logging server, or the logs can be written "directly" using standard (network) file i/o, such as NFS, to a shared logging area on a central fileserver.

At sites that already use a central fileserver for other purposes, the "direct logging to a fileserver" approach should be considered the best-practice strategy because it will be higher performance, generally more robust, and the output will be available to other tools and users in a natural way. This fileserver approach does require the shared network logfile location to be writable from every blade. The name of the share/mount as seen by each type of blade profile can be different, as dictated by the OS and mounting requirements of that class of system.

The alternate approach using the Tractor-specific logging server is the default setting, since it does not require sites to configure and mount a central fileserver on all blades. However, it does require that the logging server itself must be started and directed to the correct location for storing its log files. It also must have permissions to begin listening for connections on an unused network port on the logserver host. The command for launching the server is specified using the *SiteCmdLogServerStartup* setting in the tractor.configfile. If it is set, then tractor-engine will launch the logserver as a subprocess when the engine starts. The hostname and port choices must also be coordinated with the CmdOutputLogging settings for each blade type in blade.config.

How command output logs are retrieved

The mechanism for retrieving and viewing output logs is independent of the choice of the above method used to deliver the output files to the central log location.

The Tractor Dashboard, and other interested applications, can retrieve the log text via an HTTP request. For example, a web browser can simply fetch and display the log text directly. In order for this to work, an HTTP server is required to serve these files to requesters. This webserver can be Tractor Engine itself, or an existing web server (Apache, etc.) that is already in use at the site. It simply needs access to the logfiles through its document root scheme.

The name used for each captured output log file is chosen based on the job and task identifiers associated with the command. This allows the files to be referenced easily from user interfaces, and also to be cleaned up when no longer needed.

The filename template string can contain substitution patterns as follows:

```
%u   the job owner's login (userid, string)
%J   the job id (integer)
%T   the task id (integer)
```

Logging Command Output to a Central Fileserver

Logging Tractor command output to a central fileserver should be considered a *best-practice* technique, especially for large production sites. Here is some background on Tractor logging that will help explain the set-up steps given below.

Overall performance is the basic motivation for having each blade write its own command log output. This scheme is not as simple to manage as one where all logs automatically flow back to a common location via TCP sockets (as is case for Alfred command logs, for example). However, for very voluminous output, such as a rendering using a shader with a print statement at every pixel, writing to local disks on each blade is very fast and incurs no network traffic. Writing directly to a high-performance network share using "local" file operations can be nearly as good, and provides many additional benefits in terms of log access and management. The logging location, as seen by each blade, is defined by the *CmdOutputLogging* setting(s) in the blade. config file. More details on this setting can be found below.

When a user interface asks tractor-engine for task logs, the engine actually responds with a (JSON formatted) URL that the UI can use to fetch the logs itself. This approach lets the particular UI implementation decide how to fetch them, as well as allowing for content negotiation between the UI and the actual log server. For example, this might be important when viewing logs on a mobile device.

The form of the URL delivered from tractor-engine to the UI is determined by the *SiteCmdLogRetrievalURL* setting in the main tractor.config file.

Here is a summary the steps needed to implement centralized logging:

- Pick a fileserver with a lot of space that can be mounted as a writable directory by each blade host. Typical network filesystem schemes include NFS, DFS, and SMB.
- The selected fileserver share must be writable by the blade process owner on each blade host (not the owner of each app launched by the blade, which may be setuid to the job owner in some cases, but the owner of *the blade process* itself).
- Since the blade processes are often started at boot time as "daemons" (via init.d or launchd, or System Services on Windows), the fileserver mount usually must also be available at boot time, too, and preferably before the blade starts.
- Each blade retrieves the name of the command logging location at start-up from the central Tractor blade.config file. This file is delivered to each blade on demand by tractor-engine, and there should be only one blade.config file at a site; it contains *blade profiles* that describe various configuration settings for each *type* of blade host, including the CmdOutputLogging location.

  The name of the network share used for command logs does *not* necessarily need to be the same when seen from every blade host, but it must be the same for all blades sharing the same blade.config *profile entry*. For example, Linux machines might use an automount path while Windows machines might use a UNC path to a share. For example:

  "CmdOutputLogging": "logfile=/net/spindles/tractor/cmd-logs/%u/J%j/T%t.log"

  or

  "CmdOutputLogging": "logfile=//spindles/tractor/cmd-logs/%u/J%j/T%t.log"
- Please note that the suggested format uses forward slashes rather than backslashes, even on Windows. You can use standard windows backslash notation, however, you would need to double every backslash. The definition here is sent to the blades, and backslashes are treated as escape characters (in python).
- Note that administrators can make run-time changes to blade.config and cause the updated version to be redistributed to all running blades by sending a "reconfigure" request to the engine.
- Be sure to create the central log location before launching blades that will use it. Also do some testing to make sure that the blade owner on server hosts can create subdirectories there and write to them.
- Then decide how user interfaces should access the files in the new shared location via URL. A useful approach is to find an existing standard web server at your site that already has read access to the common log directory and can serve these (text) files on request. For example, it is possible to add a symbolic link to the log repository under the web server's main document root. (Also, see the "CORS" note, below.)
- Then change the engine's tractor.config file to reference the shared log URL. The *SiteCmdLogRetrievalURL* setting is a template for a URL that includes place-holders for the requested log file, which the engine will provide from the UI request. For example:

```
"SiteCmdLogRetrievalURL":
     "http://internal_web/links/spindles/tractor/cmd-logs/%u/J%j/T%t.log",
```

NOTE: It is very important that the logfile template '%u/J%j/T%t.log' match for both the logfile definition, and the retrieval URL.

Serving Logs via a Standard Web Server

There are several benefits to using a standard web server to deliver these command logs to Tractor Dashboard, and other requestors. The first is that it can offload this type of file i/o from Tractor Engine itself. More importantly, it can also allow users to browse logs directly from a generic web browser by simply traversing the job directory listings delivered by the web server. Obviously, the underlying shared fileserver itself provides similar access to logs from arbitrary utilities and scripts using plain file read operations.

CORS Support for "Inline" Dashboard Requests

In the Tractor Dashboard, you can view a command's logs by double clicking on the task box in the job diagram, or select "Logs" from the right-click menu for that task. This will display the logs in a new pane "within" the Dashboard interface itself. Alternatively, you can select the menu entry called "Logs (in new window)" and the logs will be displayed in a new independent browser window.

One technicality related to using a webserver rather than tractor engine to serve these log files is that it may require a small webserver configuration change to make it work successfully with your browser's "same-origin security policy" settings. These policies are useful in general for protecting your privacy in cases where javascript from one site tries to access resources from a completely different web site without your knowledge. In cases where you *d* want two websites to cooperate, such as tractor-engine and your log webserver, you need to explicitly designate the shareable content on the log webserver.

This technique is called *cross-origin resource sharing,* or **CORS** for short, and most webservers provide a way to enable it. The basic idea is that the Dashboard is made up of html and javascript content served to your browser from tractor-engine, and sometimes that javascript will make HTTP (XHR) requests to another webserver (the log server). Your browser will only allow that "cross-site" transaction if the log server allows that kind of sharing. As an example, for Apache on Linux you can add something like the following text to a new or existing file in /etc/httpd/conf.d:

```
<Directory /var/www/html/tractor-logs>
   Header set Access-Control-Allow-Origin *
</Directory>
```

You should use the actual path to your centralized log area, of course. You an also use settings that are more strict than "*" as well, please review the webserver documentation for details.