restraint Documentation

Release 0.2.0

Bill Peck, Dan Callaghan, Jeff Bastian

Jun 01, 2020

Contents

Restraint is designed to execute tasks. These tasks can be tests which report results or simply code that you want to automate. Which tasks to execute is determined by a job[1]. The job also describes where to retrieve the tasks from and what parameters to pass in. These tasks can report multiple PASS, FAIL, WARN results along with an optional score. Tasks also have the ability to report log files. Each task can have metadata describing dependencies and max run time for example. Execution and reporting can be further enhanced with plugins.

Restraint can be used with Beaker[2] since it talks Beaker's Harness API[3] for reporting results. It can also be used stand-alone.

[1] Job XML.
[2] Beaker is open-source software for managing and automating labs of test computers.
[3] Alternate Harness API.

CHAPTER 1

Features

- Tasks can be retrieved directly from git.
- Does not rely on Anaconda/kickstart to install task dependencies.
- Can be statically linked to make it easier to test the system without changing the system.
- Can be run stand-alone without Beaker.
  - Tasks are executed with the same environment (no surprises when run later in Beaker).
  - Developing tasks is much quicker since you don't have to build task RPMs, schedule a system, provision a system, etc…
- Can be easily extended with Plugins.
- Uses Beaker's job XML.

The following documentation will show you how to use Restraint in both environments.

Contents:

1.1 Installing

1.1.1 Installing from RPM

Pre-built statically linked versions are available for the following OSes:

- RedHatEnterpriseLinux
- Fedora
- CentOS

To get the appropriate repo file for your OS, use one of the commands listed below:

- RedHatEnterpriseLinux

```
# sudo wget -O /etc/yum.repos.d/beaker-harness.repo https://beaker-project.org/yum/
↪beaker-harness-RedHatEnterpriseLinux.repo
```

- Fedora

```
# sudo wget -O /etc/yum.repos.d/beaker-harness.repo https://beaker-project.org/yum/
↪beaker-harness-Fedora.repo
```

- CentOS

```
# sudo wget -O /etc/yum.repos.d/beaker-harness.repo https://beaker-project.org/yum/
↪beaker-harness-CentOS.repo
```

Once you have the appropriate repo file on your system you can install Restraint via dnf (or yum on older systems). Although you can install both the server and the client on the same machine it is not recommended.

Install the Restraint client on your machine if you want to run stand-alone jobs (i.e.: outside of Beaker):

```
# sudo dnf install restraint-client
```

Install the Restraint server on the systems that will run the tasks/tests:

```
# sudo dnf install restraint
```

1.1.2 Building from Source

Source code is located at https://github.com/beaker-project/restraint/. Restraint can be built and linked dynamically or statically. To build it dynamically you will need the development libraries for the following packages installed (minimum versions are listed):

- zlib-1.2.11
- bzip2-1.0.8
- libffi-3.1
- glib2-2.56.4
- libxml2-2.9.10
- libarchive-3.4.0
- xz-5.2.4
- libsoup-2.48.1
- intltool-0.51.0
- selinux-2.7
- curl-7.68.0
- json-c-0.13.1
- openssl-1.0.1m

Commands that will make sure most of the development libraries required are installed:

```
# sudo dnf install zlib-devel bzip2-devel libffi-devel glib2-devel libxml2-devel
# sudo dnf install libarchive-devel xz-devel libsoup-devel selinux-devel json-c-devel
# sudo dnf install intltool openssl-devel libcurl-devel
```

Once you have all the development libraries installed, you can clone Restraint from git:

```
% git clone git@github.com:beaker-project/restraint.git
% cd restraint
```

Build Restraint:

```
% make
```

To build it statically first enter the third-party directory and build the support libraries:

```
% pushd third-party
% make
% popd
```

Then build Restraint with the following command:

```
% pushd src
% PKG_CONFIG_PATH=../third-party/tree/lib/pkgconfig make STATIC=1
% popd
```

Installing Restraint:

```
% make install
```

1.2 Starting the Daemon

Regardless if you installed from RPM or from source you start the daemon one of two ways. If the system uses systemd use the following commands:

```
Enable the service for next reboot
# systemctl enable restraintd.service
Start the service now
# systemctl start restraintd.service
```

For SysV init based systems use the following commands:

```
Enable the service for next reboot
# chkconfig --level 345 restraintd on
Start the service now
# service restraintd start
```

When Restraint runs as a system service it listens on the port 8081.

1.3 Processes and Commands

There are two main restraint processes. The first is the restraint server named *restraintd* which processes tasks. The second process supports restraint standalone. This process is the restraint client named *restraint* which starts *restraintd*, provides the job.xml information to the server, and collects logs and results from the server.

1.3.1 restraintd

restraintd is the daemon which executes the tasks.

Both a SysV init script and a systemd unit file are provided. The included spec file will use the correct one when built on RHEL/Fedora based systems.

Logging messages from restraintd are printed to stderr and all output from command execution is printed to stdout.

stderr is redirected to /dev/console to help debug when things go wrong. The SysV init script redirects both stdout and stderr to /var/log/restraintd.log. For systemd, use the following *journalctl* command to view restraint logs:

```
journalctl --unit restraintd

-- Logs begin at Thu 2020-03-12 11:45:05 EDT, end at Thu 2020-03-12 12:10:47 EDT. --
Mar 12 11:45:26 virt-test systemd[1]: Starting The restraint harness....
Mar 12 11:45:26 virt-test systemd[1]: Started The restraint harness..
Mar 12 11:45:26 virt-test restraintd[1135]: recipe: * Fetching recipe: http://lc.
↪example.net:8000//recipes/30220/
Mar 12 11:45:26 virt-test restraintd[1135]: Listening on http://localhost:8081
Mar 12 11:45:26 virt-test restraintd[1135]: recipe: * Parsing recipe
Mar 12 11:45:26 virt-test restraintd[1135]: recipe: * Running recipe
Mar 12 11:45:26 virt-test restraintd[1135]: ** Fetching task: 183853 [/mnt/tests/
↪distribution/check-install]
Mar 12 11:45:26 virt-test restraintd[1135]: use_pty:FALSE rstrnt-package reinstall␣
↪beaker-core-tasks-distribution-check-install
Mar 12 11:45:32 virt-test yum[1194]: Installed: beaker-core-tasks-distribution-check-
↪install-1.0-2.noarch
Mar 12 11:45:33 virt-test restraintd[1135]: ** Preparing metadata
Mar 12 11:45:33 virt-test restraintd[1135]: ** Refreshing peer role hostnames:␣
↪Retries 0
Mar 12 11:45:33 virt-test restraintd[1135]: ** Updating env vars
Mar 12 11:45:33 virt-test restraintd[1135]: ** Updating external watchdog: 2400␣
↪seconds
Mar 12 11:45:33 virt-test restraintd[1135]: ** Installing dependencies
Mar 12 11:45:33 virt-test restraintd[1135]: ** Running task: 183853 [/distribution/
↪check-install]
...
Mar 12 11:45:43 virt-test restraintd[1135]: ** Completed Task : 183853
```

When *restraintd* runs as a system service by SysV init or systemd, it listens on the port 8081.

restraintd can also be paired with the restraint client at which case it does not run as a service. More details on *Standalone* can be found at *restraint*. In this case, any *restraintd* stdout/stderr output is directed to the *restraint* client output.

The scripts and programs associated with the *restraintd* server can be run within the context of a job as well outside a job execution.

Command Usage

Restraint commands are communicated to the running *restraintd* service by providing a URL that *restraintd* is listening to. When the command is run within a job, the needed information is available by way of environment variables set by *restraintd* for each task. When the command is executed outside a job, you can provide the information by one of three options. One option relies on setting of environment variables. Second is the server option which requires you gather restraint server port, recipe number, and task number for constructing the command URL. Lastly is a local option which relies on an environment file created by *restraintd*.

Environment Variables Option

Most often, many of restraint commands are executed in tasks included in your 'job.xml'. As a result, commands look for specific environment variables to be set by restraintd. The variables are as follows with data such as port, recipe, and task id which is unique for each job:

```
HARNESS_PREFIX=RSTRNT_
RSTRNT_URL=http://localhost:<port>
RSTRNT_RECIPE_URL=http://localhost:<port>/recipes/<recipe_id>
RSTRNT_TASKID=<task_id>
```

Note: <port> is a numeric value representing the port used to communicate with *restraintd*. <recipe_id> and <task_id> are the numeric values assigned to your running jobs recipe and task.

To utilize the environment variables option when executing a command outside your job, the command software will default to look for environment variables when other *Server* and *Local* options are not set. These environment variables must be set by the user before executing the command.

Server Option

The server option of calling these commands without exporting environment variables is to provide the argument:

```
-s, --server <server-url>
```

The format of the <server-url> is one of the following depending on the command:

```
# for rstrnt-abort
  http://localhost:<port>/recipes/<recipe_id>/status/
# for rstrnt-adjust-watchdog
  http://localhost:<port>/recipes/<recipe_id>/watchdog/
# for rstrnt-report-results
  http://localhost:<port>/recipes/<recipe_id>/tasks/<task_id>/results/
# for rstrnt-report-log. The string /logs/$file is appended by the command for you.
  http://localhost:<port>/recipes/<recipe_id>/tasks/<task_id>
```

Local Option

A simpler option is to run the command locally on the host running *restraintd* by specifying the following argument:

```
--port <server-port-number>
```

This option can be used on the same host running *restraintd* since the information is derived from the local file */var/lib/restraint/rstrnt-commands-env-<$port>.sh* (where *$port* is the port number restraintd listens on). As the server progresses through a job, it defines this file based on the current task. As a result, the user does not need to gather recipe number and task number and construct a URL for a command as this will be generated for you. The port number must be provided by the user. For *restraintd* service, the default port of 8081 can be used. When running with *restraint* client, the port number can be found in *restraint* client log output since *restraintd* output is redirected to the client. Log locations for service and non-service *restraintd* can be found in the section *restraintd*. The following log entry is the one which contains the port number of interest:

```
Listening on http://localhost:<port-number>
```

This *–port* option has similar effect to doing the following prior to executing the command:

```
export $(cat /var/lib/restraint/rstrnt-commands-env-$port.sh)
```

In conclusion, one of three methods must be used to execute your command. The following are examples of each method using the command *rstrnt-abort* as an example:

```
rstrnt-abort                                                                    #␣
↪Environment Variables method
rstrnt-abort -s http://localhost:<port>/recipes/<rid>/tasks/<tid>/status/   # Legacy␣
↪Method
rstrnt-abort --port <port>                                                      # Local␣
↪Method
```

Note:

1. Replace <port>, <rid>, <tid> with your restraint port number, recipe id, task id.
2. Given these fields change as the job progresses and if you are running the command outside the job, the window of opportunity to target the current running task is reduced when using the –port option.

rstrnt-abort

Running this command sets a recipe or a task to *Aborted* status. For an aborted recipe, the current task as well as subsequent tasks in the recipe will be marked as *aborted* and the job is discontinued.

Arguments for this command are as follows:

```
rstrnt-abort [ --port <server-port-number> ] \
               -s, --server <server-url>
             ]
```

Where:

--port <server-port-number>
: Refer to *Command Usage* for details.

-s, --server <server-url>
: Refer to *Command Usage* for details.

  Where <server-url> is as follows:

```
http://localhost:<port>/recipes/<recipe_id>/status/
```

rstrnt-adjust-watchdog

This command allows you to adjust both the external watchdog and the local watchdog.

The arguments for this command is as follows:

```
rstrnt-adjust-watchdog [ --port <server-port-number>] \
                         -s, --server <server-url>
                       ] <time>
```

Where:

--port <server-port-number>
: Refer to *Command Usage* for details.

-s, --server <server-url>
: Refer to *Command Usage* for details.

  Where server-url is *http://localhost:<port>/recipes/<recipe_id>/watchdog/*

time

This is a required argument. This time can be configured in seconds, minutes, and hours. The value of the field should be a number followed by either the letter "m" or "h" to express the time in minutes or hours. It can also be specified in seconds by giving just a number. In most cases, it is recommended to provide a value in at least minutes rather than seconds.

For example: 90 = 90 seconds, 1m = 1 minute, 2h = 2 hours

The time should be the absolute longest a test is expected to take on the slowest platform supported, plus a 10% margin of error. Setting the time too short may lead to spurious cancellations, while setting it too long may waste lab system time if the task does get stuck. Durations of less than one minute are not recommended, as they usually run some risk of spurious cancellation, and it's typically reasonable to take a minute to abort the test after an actual infinite loop or deadlock.

The time provided with the command replaces the current watchdog time as opposed to adding to or removing from the current watchdog time. Once set, it will take up to `HEARTBEAT` (1 minute) time for the local watchdog thread to wake up and see the changes (provided the metadata `no_localwatch` is false); however, the effective time is as soon as the command is executed since current time is captured. The external watchdog is increased by `EWD_TIME` (30 minutes) from the time you provide while the local watchdog uses the exact time provided.

The following log entries appear in the harness.log file as watchdog's heartbeat progresses every minute.:

```
*** Current Time: Fri May 17 15:15:49 2019 Localwatchdog at: Fri May 17 15:15:59 2019
```

When a user runs this command, you can expect to see the following log entry once the change is first recognized. Notice it is prefixed with 'User Adjusted'. Also notice in this example the expire time is less than current time. This can occur if the command was run with number of seconds less than 1 minute. There is a delay waiting for the watchdog thread to wake up to handle the changes. The thread can recognize a change occurred at a previous point in time and will expire the task immediately if the expired time is earlier than now.:

```
*** Current Time: Fri May 17 15:15:49 2019 User Adjusted Localwatchdog at: Fri May 17␣
↪15:15:02 2019
```

If the command is run with time less than the `HEARTBEAT` time, the following warning will appear when the command is executed:

```
Expect up to a 1 minute delay for watchdog thread to notice change.
```

If the task metadata has `no_localwatchdog` set to `true`, the local watchdog time is not adjusted with this new time. However, the external watchdog will continue to be adjusted. The log file will show the following warning when this occurs:

```
Adjustment to local watchdog ignored since 'no_localwatchdog' metadata is set
```

rstrnt-backup

Provides the ability to backup a list of files. This command works in concert with *rstrnt-restore* which restores the files. In order to preserve permissions and attributes of the files, it is recommended to run this command as root. The command line for this features is as follows:

```
rstrnt-backup [list of files to backup]
```

Other than the list of files to backup, there are no arguments with this command. However, there exists an environment variable which may be used:

```
RSTRNT_BACKUP_DIR - Specify an environment variable which can be set if you want
                    your files backed up in a directory other than default.
                    The default is in the subdirectory `/backup`.
```

rstrnt-package

This command supports installation, removal, and re-installation of packages for various OS package managers. The restraintd server uses the command to perform package operations for user's task *dependencies*. It may be necessary for user tasks to control these operations as part of their tests.

The arguments for this command are as follows:

```
rstrnt-package  <install | remove | reinstall> <package-name>
```

The following are environment variables available to control execution of this command:

```
RSTRNT_PKG_CMD:      To specify which package manager command to use.
                     default: yum
RSTRNT_ARG_ARGS:     To provide arguments to package manager command.
                     default: -y
RSTRNT_PKG_INSTALL:  Specify package manager install operation.
                     default: install
RSTRNT_PKG_REMOVE:   Specify package manager remove operation.
                     default: remove.
RSTRNT_PKG_RETRIES:  Number of times to retry package operation.
                     default: 5
RSTRNT_PKG_DELAY:    Number of seconds to delay between retries.
                     default: 1
```

rstrnt-prepare-reboot

Prepare the system for rebooting. Similar to rstrnt-reboot, but does not actually trigger the reboot.

If machine is UEFI and has efibootmgr installed, sets BootNext to BootCurrent and uses `NEXTBOOT_VALID_TIME` to determine for how long (in seconds) this value is valid. After the specified time, BootOrder is reset to previous state. Default value for `NEXTBOOT_VALID_TIME` is 180 seconds.

Tasks can run this command before triggering a crash or rebooting through some other non-standard means. For example:

```
rstrnt-prepare-reboot
echo c >/proc/sysrq-trigger
```

No arguments are required to run this command.

rstrnt-reboot

Helper to soft reboot the system. On UEFI systems, it will use efibootmgr to set next boot to what is booted currently. No arguments are required to run this command.

rstrnt-report-log

The command *rstrnt-report-log* loads a log file for a given task. If called multiple times for the same filename for the same task, it replaces the previously sent file.

The arguments for this command are as follows:

```
rstrnt-report-log [ --port <server-port-number> \
                    -s, --server <server-url> \
                  ] -l, --filename <logfilename>
```

Where:

--port <server-port-number>
Refer to *Command Usage* for details.

-s, --server <server-url>
Refer to *Command Usage* for details.

Where *server-url* is *http://localhost:<port>/recipes/<recipe_id>/tasks/<task_id>* *rstrnt-report-log* completes the urls by appending *logs/$file* to your server-url.

-l, --filename <logfilename>
Specify the name of log file to upload. This is a required argument.

rstrnt-report-result

The command *rstrnt-report-result* sends a result report and alters the status of the task. This command can be called multiple times for a single task each concluding with their own status results. At conclusion of the task, the final task result is the most severe rating. So if you call the command with FAIL, then WARN, then PASS, the task status results in FAIL.

This program runs in two modes. One provides backward compatibility to legacy harness and libraries and the other is restraint specific. In the latter case, there are more features. Both modes report a result file, test results, and an optional score.

Restraint Reporting Mode

For restraint reporting mode (not –rhts), the format of arguments is as follows:

```
rstrnt-report-result [--port <server-port-number>] \
                      -s, --server <server-url> \
                      -o, --outputfile <outfilename> \
                      -p, --disable-plugin <plugin-name> --no-plugins] \
                     TESTNAME TESTRESULT [METRIC]
                     ]
```

Where:

--port <server-port-number>
Refer to *Command Usage* for details.

-s, --server <server-url>
Refer to *Command Usage* for details.

Where *server-url* is *http://localhost:<port>/recipes/<recipe_id>/tasks/<task_id>/results/*

-o, --outputfile <outfilename>
Specify the name of file to upload. If not specified, the environment variable $OUTPUTFILE is used if available.

-p, --disable-plugin <plugin-name(s)>
Disables the specified reporting plugins (see *Report Result*) with the provided name or list of names. For example, to disable the built-in AVC (Access Vector Cache) checker, this argument would look like:

```
--disable 10_avc_check
```

--no-plugins
Disables all reporting plugins

TESTNAME
Testname of the task. This is a required argument.

TESTRESULT
Indicates results of job. It can be one of PASS|FAIL|WARN. This is a required argument.

METRIC
Optional result metric

Legacy Reporting Mode

The rhts extension of restraint uses –rhts. The command line would appear as follows:

```
rstrnt-report-result --rhts TESTNAME TESTRESULT LOG/OUTPUTFILE [METRIC]
```

Where:

TESTNAME
Testname of the task. This is a required argument.

TESTRESULT
Indicates results of job. It can be one of PASS|FAIL|WARN. This is a required argument.

LOGFILE
Output name of file. If not specified, the environment variable $OUTPUTFILE is used if available.

METRIC
Optional result metric

The legacy mode depends on environment variables being defined as described in *Command Usage*. The options *-s, –server* and *–port* are not supported for legacy mode.

Legacy mode looks to see if the environment variable AVC_ERROR is set to +no_avc_check. If this is true, then its behavior is equivalent to the non-legacy mode `--disable 10_avc_check` argument.

rstrnt-restore

Provides the ability to restore a previously backed up file(s). This command works in concert with *rstrnt-backup* which performs the back up step. There is a plugin which is executed at task completion which calls this command for you (*Completed* restore plugin).

rstrnt-sync-block

Block the task until the given systems in this recipe set have reached a certain state. Use this command, along with *rstrnt-sync-set* to synchronize between systems in a multihost recipe set.

```
rstrnt-sync-block -s <state> [--timeout <timeout>] [--retry <time>] [--any] <fqdn> [
↪<fqdn> ...]
```

For a more detailed guide on multihosting, refer to Beaker Multihost documentation.

-s <state>
Wait for the given state. If this option is repeated, the command will return when any of the states has been reached. This option is required.

--retry <time>
rstrnt-sync-block sleeps inbetween check for machine(s) states. If you'd like increase or decrease the frequency of checks, you can alter sleep time using the option *retry*. The default is 60 seconds.

--timeout <timeout>
Return a non-zero exit status after *timeout* seconds if the state has not been reached. By default no timeout is enforced and the command will block until either the given state is reached on all specified systems or the recipe is aborted by the local or external watchdog.

--any
Return when any of the systems has reached the given state. By default, this command blocks until *all* systems have reached the state.

<fqdn> [<fqdn> ...]
FQDN of the systems to wait for. At least one FQDN must be given. Use the role environment variables to determine which FQDNs to pass.

rstrnt-sync-set

Sets the given state for this system. Other systems in the recipe set can use *rstrnt-sync-block* to wait for a state to be set on other systems. The syntax for this command is as follows:

```
rstrnt-sync-set -s STATE
```

States are scoped to the current task. That is, states set by the current task will have no effect in subsequent tasks.

On execution of the first *set* operation, a background process *rstrnt-sync* is spawned which collects these states and responds to block requests. This server listens for events received on *TCP port 6776*. All subsequent *set* and *block* operations are forwarded to the *rstrnt-sync* server by way of this socket.

This script also writes the states to the file named */var/lib/restraint/rstrnt_events*. This file is used when the system reboots enabling the states to be restored.

1.3.2 restraint

The *restraint* client is used for standalone execution.

Use the *restraint* command to spawn a *restraintd* process to run a job on a remote test machine. You can run jobs on the local machine but it is not recommended since some tasks reboot the system. Hosts are tied to recipe IDs inside the job XML.

Arguments for the client are as follows:

```
restraint --host <recipe-id>=<host> --job <job.xml> [--restraint-path </dir/
↪restraintd>] [-v]
```

Where:

--host <recipe_id>=[<user>@]<host>
Set host for a recipe with specific id. The recipe_id identifies which host correlates to the recipe with the same recipe id in your job.xml file. This is very useful for multihost testing. If there is no id in the recipe of your job.xml file, then 1 is the default.

--job </yourdir/your-job.xml>
File Location of your job.xml.

--restraint-path </dir/restraintd>
The optional argument --restraint-path specifies the path to the restraintd binary to run on the remote machine. This can be used by developers where the restraint repo is pulled and restraintd image is built. By default, the installed image is executed.

-v
You can pass -v for more verbose output which will show every task reported. If you pass another -v you will get the output from the tasks written to your screen as well.

A sample of restraint command line is as follows:

```
restraint --host 1=addressOfMyTestSystem.example.com --job /path/to/simple_job.xml --
↪restraint-path /home/userid/restraint/src/restraintd
```

By default, the *restraintd* launched in the remote system will randomly choose a free port to listen on. The option -p, --port <port> can be used to specify the port where *restraintd* will listen on.

Restraint will look for the next available directory to store the results in. In the above example, it will see if the directory simple_job.01 exists. If it does (because of a previous run) it will then look for simple_job.02. It will continue to increment the number until it finds a directory that doesn't exist.

By default, Restraint will report the start and stop of each task run like this:

```
Using ./simple_job.07 for job run
* Fetching recipe: http://localhost:42640
* Parsing recipe
* Running recipe
*  T:   1 [/kernel/performance/fs_mark                    ] Running
*  T:   1 [/kernel/performance/fs_mark                    ] Completed: PASS
*  T:   2 [/kernel/misc/gdb-simple                        ] Running
*  T:   2 [/kernel/misc/gdb-simple                        ] Completed: PASS
*  T:   3 [restraint/vmstat                               ] Running
*  T:   3 [restraint/vmstat                               ] Completed
```

All of this information is also stored in the job.xml which in this case is stored in the ./simple_job.07 directory.

job2html.xml

An XSLT (eXtensible Stylesheet Language Transformations) template to convert the stand-alone job.xml results file into an HTML doc. The template can be found in Restraint's client directory.

Here is an example command to convert a job run XML file into an HTML doc. This HTML doc can be easily navigated with a browser to investigate results and logs.

```
xsltproc job2html.xml simple_job.07/job.xml > simple_job.07/index.html
```

job2junit.xml

An XSLT template to convert the stand-alone job.xml file into JUnit results. The template can be found in Restraint's client directory.

Here is an example command to covert a job run XML into JUnit results.

```
xsltproc job2junit.xml simple_job.07/job.xml > simple_job.07/junit.xml
```

1.3.3 Legacy RHTS Commands

Prior to the *Restraint* harness, users used *RHTS* commands in their jobs. These are being deprecated and substitutes for those legacy commands can be found in *Replacement for RHTS Scripts*.

1.4 Jobs

Restraint parses a sub-set of the Beaker job XML[1]. Here is an example showing just the elements required for running in the stand-alone configuration.

```
<job>
 <recipeSet>
  <recipe>
   <task name="/kernel/performance/fs_mark" keepchanges="yes">
    <fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git?master
↪#kernel/performance/fs_mark" />
    <params>
     <param name="foo" value="bar"/>
    </params>
   </task>
      .
      .
      .
   <task name="/kernel/foo/checker">
    <rpm name="rh-tests-kernel-foo-checker" path="/mnt/tests/kernel/foo/checker"/>
   </task>
  </recipe>
 </recipeSet>
</job>
```

1.4.1 Naming Tasks

For reporting purposes it is a good idea to name your tasks. For git tasks we have settled on a standard where we use the sub-directory path from our git repo as the task name. You can see that in the following example.

```
<task name="/kernel/performance/fs_mark">
```

This name will be used when reporting on the status of the task and when reporting results.

1.4.2 Task Roles

Restraint supports role assignment for tasks or whole recipes for use in multi-host jobs.

```
<job>
 <recipeSet>
  <recipe role="SERVERS">
```

(continues on next page)

[1] Beaker Job XML.

(continued from previous page)

```
   <task name="/kernel/filesystems/nfs/connectathon-mh">
    <fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git?master
↪#kernel/filesystems/nfs/connectathon-mh" />
   </task>
  </recipe>
  <recipe>
   <task name="/kernel/filesystems/nfs/connectathon-mh" role="CLIENTS">
    <fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git?master
↪#kernel/filesystems/nfs/connectathon-mh" />
   </task>
  </recipe>
 </recipeSet>
</job>
```

The above example results in environment variables “SERVERS” and “CLIENTS” containing hostnames assigned to corresponding recipes. The variables will be available only to tasks with the same padding within recipes.

Recipe roles function as default roles for tasks that have no role specified and can be overridden by task roles.

Apart from role env variables Restraint also exports 2 more hostname-related variables:

- RECIPE_MEMBERS - contains hostnames of all hosts within current recipeSet.
- JOB_MEMBERS - contains hostnames of all hosts in current job.

1.4.3 Keeping Your Task Changes Intact

By default Restraint will fetch tasks every time you run a recipe overwriting any changes you’ve done locally. This is not desirable in some cases, e.g. when debugging a test. Restraint provides the ability to keep local changes by setting task property “keepchanges” to “yes” in the job xml.

```
<task name="/kernel/performance/fs_mark" keepchanges="yes">
```

1.4.4 Installing Tasks

The above example shows that you can install tasks directly from git or from an RPM in a yum repo.

Fetch

The first example shows fetching a task from git.

```
<fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git?master
↪#kernel/performance/fs_mark" />

OR

<fetch ssl_verify="off" url="https://fedorapeople.org/cgit/bpeck/public_git/tests.git/
↪snapshot/tests-master.tar.gz#kernel/performance/fs_mark" />
```

The fetch node accepts git URI’s that conform to the following:

- Prefixed with git:// OR use tarballs with http:// and cgit can serve them automatically.
- The fully qualified hostname. Remember that the system running restraintd must be able to reach this host.

- The path to the git repo.
- Optionally you can specify a valid reference which can be a branch, tag or SHA-1. ie: ?master
- Optionally you can specify a sub-dir. Restraint will only extract this sub-dir and run the task from here. ie: #kernel/performance/fs_mark. Notice that there is not a preceding slash here.
- If you need to disable SSL certificate checking you can set ssl_verify parameter to "off".

Restraint uses git's archive protocol to retrieve the contents so make sure your git server has enabled this. You can enable this on most servers by putting the following in your git repo config

```
[daemon]
       uploadarch=true
```

RPM

The second example will attempt to install the task via yum/rpm.

```
<rpm name="rh-tests-kernel-foo-checker" path="/mnt/tests/kernel/foo/checker"/>
```

Currently Restraint does not attempt to set up any repos that you may have specified in your job.xml. This means that in order for it to install the above task you must have already configured the task repo on the machine running restraintd.

The path attribute tells restraint where the task scripts are installed.

1.4.5 Parameters

You can optionally pass parameters to a task via environment variables. The following snippet from our example would create an environment variable named 'foo' with the value 'bar'.

```
<params>
 <param name="foo" value="bar"/>
</params>
```

The parameter KILLTIMEOVERRIDE allows you to specify a different max time than what is specified in the tasks metadata. KILLTIMEOVERRIDE is provided for compatibility with legacy RHTS (Red Hat Test System).

As of 0.1.40, the parameter RSTRNT_MAX_TIME has been deprecated in favor of KILLTIMEOVERRIDE because of confusion with RSTRNT_MAXTIME

The parameter RSTRNT_USE_PTY allows you to either enable or disable using a pty for task execution. Use `true` to enable and `false` to disable. Setting this value in the job will override the settings in metadata or testinfo.desc.

1.5 Tasks

Restraint doesn't require tasks to be written in any particular language. In fact, most tests are written in a mixture of shell, python and C code. You do need to provide some metadata in order for things to work best.

1.5.1 Restraint Metadata File

Restraint will look for a file called metadata in the task directory. The format for that file is a simple *ini* file which most people should be familiar with.

```
[General]
name=/restraint/env/metadata
owner=User ABC1 <userabc1@example.com>
description=just reports env variables
license=GPLv2
confidential=no
destructive=no

[restraint]
entry_point=./runtest.sh
max_time=5m
dependencies=gcc;emacs
softDependencies=numactl;numactl-devel
environment=META_VAR1=var1value;META_VAR2=var2value;META_VAR3=var3value
repoRequires=general/include;filesystems/include
no_localwatchdog=true
use_pty=false
```

restraintd does not require any metadata fields to be present. In other words, there are no checks and reporting of errors if metadata is not present. This allows flexibility in your configuration.

The *General* section is mostly used for informational data. The only element that *Restraint* will process is the *name* attribute. If defined, this will overwrite the task name specified from the job XML.

The *restraint* section has the following elements which can be defined:

entry_point

This tells Restraint how it should start running the task. If you don't specify a program to run it will default to 'make run' which is what legacy RHTS (Red Hat Test System) would do. This would require you provide a Makefile. Other examples of entry points:

```
* entry_point=autotest-local control-file
* entry_point=STAF local PROCESS START SHELL COMMAND "ps | grep test | wc >testcount.
↪txt"
```

max_time

The maximum time a task is expected to run. When restraintd runs a task it sets up a localwatchdog which will kill the task after this time has expired. When run in Beaker this is also used for the external watchdog (typically 20-30 minutes later than the local watchdog time). Time units can be specified as follows:

```
* d for days
* h for hours
* m for minutes
* s for seconds
```

To set a max run time for 2 days you would use the following:

```
max_time=2d
```

dependencies

A semicolon-delimited (`;`) list of additional packages (needed to run this task) to be installed on the system. The task will abort if the dependencies fail to install.

```
dependencies=lib-virt;httpd;postgresql;nfs-utils;net-tools;net-snmp;ethereal;
↪wireshark;tcpdump;rusers;bzip2;gcc
```

environment

A semicolon-delimited (`;`) list of task environment variables to be set on the system.

```
environment=META_VAR1=var1value;META_VAR2=var2value;META_VAR3=var3value
```

softDependencies

A semicolon-delimited (`;`) list of optional additional packages to be installed on the system. The task will proceed even if the soft dependencies fail to install. This is useful for a task that is intended to run on multiple platforms, and the task can test platform-specific features (e.g., NUMA) if the appropriate support packages are installed, but the task will not abort on the other platforms where the support packages do not exist.

```
softDependencies=numactl;numactl-devel
```

repoRequires

A semicolon-delimited (`;`) list of additional tasks needed for this task to run.

```
repoRequires=general/include;filesystems/include
```

Note: When fetching from git (see *Fetch*), this is the `#subdirectory` portion of the URL, so do *not* use a leading `/` character as was done with RhtsRequires in testinfo.desc for Legacy RHTS tasks.

no_localwatchdog

Normally Restraint will setup a localwatchdog which will attempt to recover from a hung task before the external watchdog (if running under Beaker) triggers. But you can tell Restraint to not setup a localwatchdog monitor by including this key with a value of `true`. Only `true` or `false` are valid values.

```
no_localwatchdog=true
```

use_pty

Before version 0.1.24 Restraint would execute all tasks from a pty. This meant that programs thought they were running in an interactive terminal and might produce ANSI codes for coloring and line positioning. Now the default is not to use a pty which will give much cleaner output. If you find your test is failing because it expects a pty you can enable the old behavior by setting this.

```
use_pty=true
```

OSMajor Specific Options

Any of the above elements can be overridden with OSMajor specific options. In order for this to work the OSMajor (or "OS family") attribute must be filled in the job.xml. If the job was run through Beaker this will have been filled in for you. If you run a stand-alone job (with restraint-client) you can set the value in the family attribute of the recipe tag. For example:

```
<job>
  <recipeSet>
    <recipe family="RedHatEnterpriseLinuxServer5">
      ...
```

For example, if a task is known to take twice as long on RedHatEnterpriseLinuxServer5 then you could use following:

```
max_time=5m
max_time[RedHatEnterpriseLinuxServer5]=10m
```

Another example where we will install RHDB on RedHatEnterpriseLinuxServer5 and PostgreSQL on everything else.

```
dependencies=postgresql
dependencies[RedHatEnterpriseLinuxServer5]=rhdb
```

1.5.2 Legacy Metadata File

Prior to the *Restraint* harness, users defined *testinfo.desc* file as the metadata file in their job tasks and restraint supported that file. This is being deprecated and the substitute for this file and variables within can be found in *Replacement for RHTS testinfo.desc File*.

1.6 Task Environment Variables

Restraint exports the following environment variables for task use. They can be altered using the environment variable of the *metadata* file or *testinfo.desc* file (see *Tasks*).

Restraint Variables	Description	Source
HOME	home directory defaults to /root. Can be overwritten using recipe or task params.	Static
HOSTNAME	Set by task plugin before execution of user task	Task Plugin
LANG	Environment variable to specify locale. The default is *en_US.UTF-8*. It can be overwritten using recipe or task params.	Static
PATH	Program search path environment variable. The default default is "/usr/local/bin:/usr/bin:/bin: /usr/local/sbin:/usr/sbin:/sbin". It can be overwritten using recipe or task params.	Static
RSTRNT_JOBID	Populated from the job_id attribute of the recipe node.	Job
RSTRNT_MAXTIME	Maximum time in seconds for this task to complete. Input to local and external watchdog timers.	Job
RSTRNT_OSARCH	OS Architectures. Ex: x86_64, s390x, i386, aarch64, ppc64, ppc64le, armhfp	Job/Task Plugin
RSTRNT_OSDISTRO	Name of the distro (Provided if running in Beaker).	Job
RSTRNT_OSMAJOR	OS Major Version of Distro. Ex: Fedora31, CentOS7, RedHatEnterpriseLinux8	Job/Task Plugin
RSTRNT_OSVARIANT	Some distros use variants. Ex: Server, Client	Job
RSTRNT_OWNER	Populated from the owner attribute of the job node.	Job
RSTRNT_REBOOTCOUNT	The number of times the system has rebooted for this task. If no reboot occurred, the values is 0.	Restraint
RSTRNT_RECIPEID	Populated from the id attribute of the recipe node.	Job
RSTRNT_RECIPESETID	Populated from the recipe_set_id attribute of the recipe node.	Job
RSTRNT_TASKID	Populated from the id attribute of the task node.	Job
RSTRNT_TASKNAME	Name of task from the job. Ex: "/distribution/command".	metadata
RSTRNT_TASKORDER	Sequence Order of tasks multiplied by 2. Used by Restraint when it performs multihosting.	Restraint
RSTRNT_TASKPATH	Path the task is installed.	rpm path/ Restraint
TERM	Terminal type defaults to *vt100*. Can be overwritten using recipe or task params.	Static
TESTID	Contains the ID assigned to this task.	Job

For legacy RHTS variables, refer to *Legacy RHTS Task Environment Variables*.

1.7 Script/Plugin Environment Variables

This table lists environment variables which affect outcome of restraint scripts and plugins. These variables are often set by the user. They are as follows:

Restraint Variables	Description	Source
AVC_ERROR	Refer to *Legacy Reporting Mode* for replacement.	User
FAILURESTRINGS FALSESTRINGS	Used by report_result plugin to report user's task. Details can be found *Report Result*	User
CLIENTS, SERVERS, DRIVERS	Assist in the execution of the scripts rstrnt-sync-block/set. *rstrnt-sync-block*	User
NEXTBOOT_VALID_TIME	Assist in the execution of the script rstrnt-prepare-reboot. *rstrnt-prepare-reboot*	Default/ User
OUTPUTFILE	Used by localwatchdog plugin to report user's task output if set.	User
TESTPATH/logs2get	File used by localwatchdog plugin to log user's files listed in logs2get.	User
RSTRNT_BACKUP_DIR	To specify directory when using using Restraint's backup/restore scripts. *rstrnt-backup*	User
RSTRNT_DISABLED	User populated to disable a plugin from running. Do *RSTRNT_DISABLED="99_reboot"* to prevent *99_reboot* from running after local watchdog expires. Do *RSTRNT_DISABLED="01_dmesg_check 10_avc_check"* to prevent multiple error checking plugins from running (though disabling these is not advised).	User
RSTRNT_DISABLE_LINGER	Used by task_run plugin to disable user lingering. Refer to OS command loginctl enable/disable linger for details. This was introduced due to behavior changes from Fedora24+. Default is to enable.	User
RSTRNT_LOGGING	Enables debugging for plugins. Default: 3 (1=Debug, 2=Info, 3=Warning, 4=Error, 5=Critical)	User
RSTRNT_NOPLUGINS	Set by restraint to disable some plugin functionality when "task_run" plugins execute. Further details on this variable can be found *Plugins*.	Restraint
RSTRNT_PKG_CMD RSTRNT_PKG_ARGS RSTRNT_PKG_INSTALL RSTRNT_PKG_REMOVE RSTRNT_PKG_RETRIES RSTRNT_PKG_DELAY	These variables are used to control the behavior of the command rstrnt-package. For more details, refer to TBD Make reference to rstrnt-package	Default/ User
RSTRNT_PLUGINS_DIR	Specifies the directory to run localwatchdog or report_result plugins.	Restraint

1.8 Plugins

Restraint relies on plugins to execute tasks in the correct environment and to check for common errors or simply to provide additional logs for debugging issues. Here is a typical outline of how plugins are executed:

```
run_task_plugins
 \
  10_bash_login
  |
  15_beakerlib
  |
  20_unconfined
  |
  25_environment
  |
  make run
```

(continues on next page)

(continued from previous page)

```
  |\
  | report_result
   \
    report_result

run_task_plugins
 \
  10_bash_login
  |
  15_beakerlib
  |
  20_unconfined
  |
  25_environment
  |
  run_plugins <- completed.d
  \
   98_restore
```

The report_result commands above cause the following plugins to be executed:

```
run_task_plugins
 \
  05_linger
  |
  10_bash_login
  |
  15_beakerlib
  |
  20_unconfined
  |
  25_environment
  |
  30_restore_events
  |
  35_oom_adj
  |
  run_plugins <- report_result.d
   \
    01_dmesg_check
    |
    10_avc_check
    |
    20_avc_clear
    |
    30_dmesg_clear
```

These plugins do not run from the task under test. They run from restraintd process. This allows for greater flexibility if your task is running as a non-root user since a non-root user would not be able to inspect some logs and wouldn't be able to clear dmesg log.

1.8.1 Task Run

Task run plugins are used to modify the environment under which the tasks will execute. Simply place the executable in /usr/share/restraint/task_run.d. The list of files in this directory will be passed to exec in alphabetical order.

Restraint currently ships with two task run plugins:

- 05_linger - Enables session bus for user that Restraint is running as. You can disable this with RSTRNT_DISABLE_LINGER=1
- 10_bash_login - invoke a login shell.
- 15_beakerlib - Sets env vars to tell beakerlib how to report results in Restraint.
- 20_unconfined - if selinux is enabled on system run task in unconfined context.
- 25_environment - Will attempt to guess certain variables if they weren't defined, (OSARCH, OSMAJOR, etc..).
- 30_restore_events - Restores Multi-host states after a reboot.
- 35_oom_adj - sets the oom score low so we are less likely to be killed.

So the above plugins would get called like so:

```
exec 05_linger 10_bash_login 15_beakerlib 20_unconfined 25_environment 30_restore_
↪events 35_oom_adj "$@"
```

In order for this to work the task run plugins are required to exec "$@" at the end of the script. Although task run plugins can't take any arguments they can make decisions based on environment variables.

It should be pointed out that the task run plugins are executed for all other plugins! This is to ensure plugins run with the same environment as your task. When executed under all other plugins the following variable will be defined:

```
RSTRNT_NOPLUGINS=1
```

You can do conditionals based on this so lets create a plugin which will start a TCP capture:

```
# Capture tcpdump data from every task
cat << "EOF" > /usr/share/restraint/plugins/task_run.d/30_tcpdump
#!/bin/sh -x

rstrnt_info "*** Running Plugin: $0"

# Don't run from PLUGINS
if [ -z "$RSTRNT_NOPLUGINS" ]; then
  tcpdump -q -i any -q -w $RUNPATH/tcpdump.cap 2>&1 &
  echo $! > $RUNPATH/tcpdump.pid
fi

exec "$@"
EOF
chmod a+x /usr/share/restraint/plugins/task_run.d/30_tcpdump
```

Refer to section (*Completed*) for how to report these results.

1.8.2 Report Result

Every time a task reports a result to Restraint these plugins will execute.

- 01_dmesg_check - This plugin checks dmesg output for lines containing certain values and also allows lines to be omitted. If any lines are selected, this indicates an error so the task will conclude with failed results.
- 30_dmesg_clear - This plugin clears dmesg log so the next task will start with a fresh log.

There are 2 variables which manage selection of dmesg output. They are *FAILURESTRINGS* and *FALSESTRINGS*. The *FAILURESTRINGS* variable contain values which allow you to select those lines considered in error. The *FALSESTRINGS* variable contain values allowing you to omit some lines. This enables you to omit false positives.

There are 3 ways *FAILURESTRINGS* and *FALSESTRINGS* configuration are provided. They can be configured by way of environment variables, as files, or defaults. The order of precedence for these variables/files is follows:

```
1) Task environment variable
2) User defined files
3) and defaults.
```

FAILURESTRINGS and *FALSESTRINGS* are processed separately so you could define *FAILURESTRINGS* as an environment variable while maintaining defaults for *FALSESTRINGS*.

The default values for *FAILURESTRINGS* are as follows:

```
Oops|BUG|NMI appears to be stuck|Badness at
```

The default values for *FALSESTRINGS* are as follows:

```
BIOS BUG|DEBUG|mapping multiple BARs.*IBM System X3250 M4
```

Both of the above strings can be overridden for each task by passing in your own *FAILURESTRINGS* or *FALSESTRINGS* environment variables. This is configured for each task. To define environment variables, refer to instructions for *metadata* or *testinfo.desc* files in (see *Tasks*).

If you want all tasks in a recipe to use the same set of your user-defined *FAILURESTRINGS* or *FALSESTRINGS*, you could start your recipe with a task which creates the following files respectively:

```
/usr/share/rhts/failurestrings
/usr/share/rhts/falsestrings
```

When configuring these files, each string should be on a separate line instead of separated with 'l'. For example, failurestrings would contain something like the following:

```
Oops
BUG
NMI appears to be stuck
Badness at
```

In some cases, the kernel will produce a multi-line error message (including hardware information and stack trace) in the dmesg output which is delimited by a "cut here" line at the beginning and an "end trace" line at the end. This plugin will capture the entire contents of the multi-line trace and considers it as a single failure. The FALSESTRINGS pattern is applied to the whole trace to check for false positives.

- 10_avc_check - This plugin searches for AVC (Access Vector Cache) errors that have occurred since the last time a result was reported.
- 20_avc_clear - This moves the time stamp used by avc_check forward so that we don't see the same AVC's reported again, some tests might generate AVC's on purpose and disable the check but you will still want to move the time stamp forward.

If you need to skip error checking, refer to RSTRNT_DISABLED as described in the *Task Environment Variables* section.

1.8.3 Local Watchdog

These plugins will only be executed if the task runs beyond its expected time limit. Restraint currently ships with three plugins:

- 10_localwatchdog - uploads the resultoutputfile.log of the running task.
- 20_sysinfo - Collects and uploads system information.
  - Uploads system log which contains a collection of system information such as slabinfo, list of blocked tasks derived from `sysrq m, t and w`, and pre-existing system log messages. Depending if journalctl exists, file journalctl or /var/log/messages is uploaded.
  - Uploads ps-lwd.log which contains a verbose list of running processes.
  - Uploads dmesg log if it contains any output.
  - Uploads user logs listed in $TESTPATH/logs2get.
- 99_reboot - Simply reboots the system to try and get the system back to a sane state. If you need to skip this step, you can use RSTRNT_DISABLED as described in (see *Task Environment Variables*).

1.8.4 Completed

These plugins will get executed at the end of every task, regardless if the localwatchdog triggered or not. The only plugin currently shipped with Restraint is:

- 85_sync_multihost_tasks - Synchronizes tasks between client/server jobs on multihost machines. This will synchronize only if there exists recipes with role=SERVERS as well as role=CLIENTS. For further details on this feature, refer to Beaker Multihost documentation[1].
- 97_audit_rotate - Searches log files in audit directory to find avc messages.
- 98_restore - Restores files backed up by either rhts-backup or rstrnt-backup.

To finish our tcpdump example from above we can add the following:

```
#Kill tcpdump and upload
cat << "EOF" > /usr/share/restraint/plugins/completed.d/80_upload_tcpdump
#!/bin/sh -x

kill $(cat $RUNPATH/tcpdump.pid)
rstrnt-report-log -l $RUNPATH/tcpdump.cap
EOF
chmod a+x /usr/share/restraint/plugins/completed.d/80_upload_tcpdump
```

If you need to skip file restoration, refer to RSTRNT_DISABLED as described in the environment variable section (see *Task Environment Variables*).

1.9 Using Restraint

1.9.1 Running in Beaker

Beaker will use restraint by default if you are running Red Hat Enterprise Linux version 8 or later or if you are running Fedora.

[1] Beaker Multihost documentation.

To use Restraint in Beaker for earlier versions of Red Hat Enterprise Linux or Fedora, you will need to specify 'restraint' as the harness:

```
<recipe ks_meta="harness=restraint">
<repos>
 <repo name="restraint"
       url="https://beaker-project.org/yum/harness/CentOS7/"/>
</repos>
 .
 .
 .
</recipe>
```

If you have tasks/tests that were written for legacy RHTS (Red Hat Test System) you can install the restraint-rhts sub-package which will bring in the legacy commands so that your tests will execute properly. Some tasks/tests have also been written with beakerlib. Here is an example recipe node that will install both for you:

```
<recipe ks_meta="harness='restraint-rhts beakerlib'">
.
.
.
</recipe>
```

If you are using Beaker command line workflows use these command line options:

```
bkr <WORKFLOW> --ks-meta="harness=restraint" --repo https://beaker-project.org/yum/
↪harness/CentOS7/
```

If you need RHTS compatibility and/or beakerlib you can add it here as well:

```
bkr <WORKFLOW> --ks-meta="harness='restraint-rhts beakerlib'" --repo https://beaker-
↪project.org/yum/harness/CentOS7/
```

1.9.2 Running Standalone

Restraint can run on its own without Beaker, this is handy when you are developing a test and would like quicker turn around time. Before Restraint you either ran the test locally and hoped it would act the same when run inside Beaker or dealt with the slow turn around of waiting for Beaker to schedule, provision and finally run your test. This is less then ideal when you are actively developing a test.

You still need a job XML file which tells Restraint what tasks should be run. Here is an example where we run three tests directly from git:

```
<?xml version="1.0"?>
<job>
  <recipeSet>
    <recipe id="1">
      <task name="/kernel/performance/fs_mark">
        <fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git?
↪master#kernel/performance/fs_mark"/>
      </task>
      <task name="/kernel/misc/gdb-simple">
        <fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git?
↪master#kernel/misc/gdb-simple"/>
      </task>
      <task name="/kernel/standards/usex" role="None">
```

(continues on next page)

(continued from previous page)

```
      <fetch url="git://fedorapeople.org/home/fedora/bpeck/public_git/tests.git
↪#kernel/standards/usex"/>
    </task>
  </recipe>
 </recipeSet>
</job>
```

Tell Restraint client to run a job:

```
restraint --job /path/to/job.xml
```

You probably don't want to run the restraintd server on the machine you use for day to day activity. Some tests can be destructive or just make unfriendly changes to your system. Restraint client allows you to run tasks on a remote system. This means you can have the task git repo on your development workstation and verify the results on your test system. In order for this to work your git repo and the recipe XML need to be accessible to your test system. Be sure to have the restraint-client package installed on the machine you will be running the restraint client command from

Here is an example:

```
restraint --host 1=addressOfMyTestSystem.example.com --job /path/to/job.xml --
↪restraint-path /home/userid/restraint/src/restraintd -v
```

This will spawn the restraintd server from the path specified in `--restraint-path` on host addressOfMyTestSystem.example.com and tell it to run the recipe with id="1" from this machine. Also remember that the tasks which are referenced inside of the recipe need to be accessible a well. Here is the output:

```
restraint --host 1=addressOfRemoteSystem --job simple_job.xml --restraint-path /home/
↪userid/restraint/src/restraintd -v
Using ./simple_job.07 for job run
* Fetching recipe: http://192.168.1.198:8000/recipes/07/
* Parsing recipe
* Running recipe
*  T:   1 [/kernel/performance/fs_mark                    ] Running
**      1 [Default                                        ] PASS
**      2 [Random                                         ] PASS
**      3 [MultiDir                                       ] PASS
**      4 [Random_MultiDir                                ] PASS
*  T:   1 [/kernel/performance/fs_mark                    ] Completed: PASS
*  T:   2 [/kernel/misc/gdb-simple                        ] Running
**      5 [/kernel/misc/gdb-simple                        ] PASS Score: 0
*  T:   2 [/kernel/misc/gdb-simple                        ] Completed: PASS
*  T:   3 [/kernel/standards/usex                         ] Running
**  :   6 [/kernel/standards/usex                         ] PASS
*  T:   3 [/kernel/standards/usex                         ] Completed: PASS
```

All results will be stored in the job run directory which is 'simple_job.07' for this run. In this directory you will find 'job.xml' which has all the results and references to all the task logs. You can convert this into HTML with the following command:

```
xsltproc job2html.xml simple_job.07/job.xml >simple_job.07/index.html
```

`job2html.xml` is found in Restraint's `client` directory.

1.9.3 Running in Beaker and Standalone

Sometimes the tests that I am developing can be destructive to the system so I don't want to run them on my development box. Or the test is specific to an architecture so I can't use a VM for it on my machine. These are cases where it's really handy to use a combination of Beaker for provisioning and Standalone for executing the tests. By default, Beaker provides a test harness for all imported distributions. You can replace test harness with your build by adding a new repository. You can create your build on your own or you can use different RPM build systems, for example COPR. Be aware that custom restraint should have higher NVR than the latest released version and your build needs to be built against distribution you planning to test. Otherwise, DNF may pick up Restraint provided by Beaker or Restraint may fail to install.

First step is to run the following workflow to reserve a system in Beaker:

```
<job><whiteboard>restraint reservesys</whiteboard>
 <recipeSet>
  <recipe ks_meta="harness=restraint" id="1">
   <distroRequires>
    <and>
       <distro_family op="=" value="Fedorarawhide"/>
       <distro_variant op="=" value="Everything"/>
       <distro_name op="=" value="Fedora-Rawhide-20200406.n.0"/>
       <distro_arch op="=" value="ppc64le"/>
    </and>
   </distroRequires>
   <hostRequires/>
   <repos>
    <repo name="my_custom_restraint" url="http://copr-be.cloud.fedoraproject.org/path/
↪to/copr/repo/results"/>
   </repos>
   <task name="/distribution/check-install" role="STANDALONE" />
   <task name="/distribution/reservesys" role="None">
    <fetch url="https://github.com/beaker-project/beaker-core-tasks/archive/master.zip
↪#reservesys"/>
   </task>
  </recipe>
 </recipeSet>
</job>
```

This will reserve a ppc64 system running Fedora Rawhide. The /distribution/reservesys task will email the submitter of the job when run so you know the system is available. By default the reservesys task will give you access to the system for 24 hours, after that the external watchdog will reclaim the system. You can extend it using extendtesttime.sh on the system.

You can spawn a second instance of restraintd server using the client command below. It will generate an instance with a different port than the port used by beaker.

```
restraint --host 1=FQDN.example.com --job simple_job.xml --restraint-path /home/
↪userid/restraint/src/restraintd -v
```

If you want to run restraint commands such as `rstrnt-adjust-watchdog nn` or `rstrnt-abort` against this test set-up, you must first export the environment variables which includes the dynamically created communication port. To do this, run the following:

```
export $(cat /etc/profile.d/rstrnt-commands-env.sh)
```

If the task you are developing doesn't work as expected you can make changes and try again. Just remember to push your changes to git, the system under test will pull from the git URL you put in your job XML.

1.10 Release Notes

1.10.1 Restraint 0.2.2

1.10.2 Bug Fixes

- restraint client now honors recipe params as well as task params.
- Correct commands exit status when argument parsing fails due to bad syntax. Commands always return non-zero in case of failure.
- Resolve loop in local watchdog plugin

  When the local watchdog (LWD) expires a task, the LWD plugin *20_sysinfo* goes into an infinite loop since the directory */mnt/testarea* is not created for the non-rhts restraint package. An error returned by *diff* utility within an infinite loop was not anticipated. The fix terminates the infinite loop when diff returns error.

1.10.3 Restraint 0.2.1

What's New

- Add ability to select *restraintd* instance by port to *restraint* commands
  When running commands outside of jobs on the local host, some *restraint* commands require manually setting up environment variables or constructing long URLs before running. This can be issue if you are trying to extend the watchdog in a timely fashion. A new option was added which requires the argument *–port <restraint-port-number>*. Commands affected are *rstrnt-report-log*, *rstrnt-report-result*, *rstrnt-abort*, and *rstrnt-watchdog*.
- Restore ability to specify *restraintd* port
  Add the *-p, –port* option back to restraint daemon and client to specify the port where *restraintd* will be listening to. RHBZ#1821342
- Document how to remove RHTS from Jobs
  Added new section *Guide to removing RHTS from Jobs* detailing substitutes for *RHTS* scripts, environment variables, and testinfo.desc file and associated variables. RHBZ#1802610

Bug Fixes

- Redirect task STDIN back to /dev/null
  In release 0.2.0, the task STDIN was redirected to a pipe shared with the server. This breaks *ausearch* command when the input is not explicitly specified, as by default, if STDIN is a pipe, it will read from it, instead of system logs. As the pipe is closed when the task is running, tests expecting matches failed, and tests expecting no matches were unreliable. Restoring redirect of task STDIN back to /dev/null ensures that *ausearch* reads from system logs by default.
- Restore default port for restraind system service
  In release 0.2.0 the port for *restraintd* system service is chosen dynamically, breaking workflows where the port was expected to persist between reboots. When *restraintd* runs as a system service, the port defaults to *8081*. RHBZ#1823545
- Restraintd killed by SIGTRAP
  It was discovered that an error logging function (g_error) introduced in 0.2.0 also performed aborts. The function was replaced with one which logs without undesirable side effects. RHBZ#1823840, RHBZ#1831824

- *restraintd* fails to start if both, IPv4 and IPv6, are not available on the loopback interface
  In this release, restraintd will not fail if it's able to listen on at least one protocol, IPv4 or IPv6, although it will still try to listen on both.
- Fix use of uninitialized FD for STDIN when PTY is requested
  When PTY was requested, the FD for the task STDIN was left uninitialized. The value, set to 0, was still used in a close call, closing the parent STDIN FD and causing unexpected behavior in task execution. In this release, the FD for STDIN is not used when PTY is requested.

1.10.4 Restraint 0.2.0

Upgrades

- RHBZ#1667510: Remove libssh from restraint client.
  The port used by restraint server is no longer static. If using the restraint client, refer to restraint documentation for changes to arguments passed since the port is no longer included in *–host* argument. The client spawns restraintd for you so the extra step of starting up a restraintd instance is no longer needed. Because of these interface changes, the restraint client and server must be the same version.
  (Contributed by Bill Peck and Carol Bouchard)
- RHBZ#1770230: Replace rhts-sync- with rstrnt-sync- cmds.
  This changeset creates rstrnt-sync- commands and links rhts-sync- commands to it. The multihost plugin now uses rstrnt-sync- commands.
  (Contributed by Carol Bouchard)
- RHBZ#1802261: Upgrade libxml2 to version 2.9.10
  (Contributed by Daniel Rodriguez Gonzalez)

Bug Fixes

- RHBZ#1795915: Remove execute permission from systemd service file. There is a warning message in the systemd logs about the file being executable.
  (Contributed by John Villalovos)

1.10.5 Restraint 0.1.45

- FIXED: RHBZ#1795781: Multihost sync hangs on remote reboot. Users multihost synchronization task hangs on block operation when remote host reboots. This is a corner case difficult to reproduce.
  (Contributed by Carol Bouchard)
- FIXED: RHBZ#1792466: Restraint segfault during labcontroller timeout. On error when gathering peer roles from the lab controller, a double free of the error structure causes bad behavior in glib memory management. Eventually this causes restraint server to crash on a segfault.
  (Contributed by Carol Bouchard)
- FIXED: RHBZ#1691485: Rstrnt Client not provide task vers in job.xml. This change affects rpm tasks only. Restraint server gets the version number from the rpm and returns it in 'Completed/Aborted' status message sent to restraint client. The restraint client writes it out in the job.xml.
  (Contributed by Carol Bouchard)
- FIXED: RHBZ#1793114: Wrong file permission on 30_dmesg_clear plugin. The new 30_dmesg_clear plugin does not have execute file permission. However, other scripts add execution permission so it is correct in the rpm. This is being fixed in repo to prevent chasing it as an issue.

(Contributed by Carol Bouchard)

1.10.6 Restraint 0.1.44

- FIXED: RHBZ#1788252: restraintd crash in timeout_callback functions. Ran into timing issues when process_timeout_callback occurs after process_pid_callback. The task data is NULL so process_timeout_callback should not attempt to process task data when pid is 0 indicating process is complete.
  (Contributed by Carol Bouchard)
- FIXED: RHBZ#1781722: Not executing task when multihost utilized. Observed that restraint reported the task started but output from the task itself not making it to taskout.log file. With debug enabled, found it stopped in 30_restore_events plugin. Performed more detail unit testing on rstrnt-sync and resolved a number of issues found.
  (Contributed by Carol Bouchard)
- FIXED: RHBZ#1782422: Fetch https operation noisy harness.log. When using <fetch url="https://github.com/repo#dirname> in task, the entire repo is downloaded and a log entry for each file/dir found is logged. These log entries get reported to Lab Controller which results in reduced performance. Fixed code to report only entries found beneath the directory name 'dirname'.
  (Contributed by Carol Bouchard)

1.10.7 Restraint 0.1.43

- FIXED: RHBZ#1774211: Seeing too many repo extraction. Under certain conditions, restraint was failing to go to next repoRequires operation causing redundant fetch operations to occur.
  (Contributed by Carol Bouchard)
- FIXED: RHBZ#1236568: Separate dmesg clear from check. Need for a separate plugin so clear of the dmesg logs is done independently from check dmesg logs. Currently this is done during *dmesg check* plugin. If *dmesg check* plugin is disabled, so is the clear operation leaving the next task will process unrelated errors. By separating clear from check operation, the clear operation can always be performed.
  (Contributed by Carol Bouchard)
- FIXED: RHBZ#1749316: Rstrnt retry refresh role on socket io err. User periodically observed "Error: Socket I/O Timed out". This occurred during the restraint task state "** Refreshing peer role hostnames" which collects host roles from lab controller and there is no response in default 1 minute time frame. To handle network issues, restraint will retry this event similar to what is done when performing fetch operations.
  (Contributed by Carol Bouchard)
- FIXED: RHBZ#1762731: Rstrnt add more metadata UTs.
  (Contributed by Carol Bouchard)
- NEW: RHBZ#1455763: New command rstrnt-prepare-reboot. It does the same preparatory work as rstrnt-reboot, but does not trigger the reboot. Tasks can use this prior to (intentionally) crashing the system or rebooting it in some other non-standard way.
  (Contributed by Tomas Klohna)

1.10.8 Restraint 0.1.42

- FIXED: RHBZ#1753652: Multihost Sync Improvements. A number of improvements have been made to the Multihost synchronization feature. * Only perform multihost sync when roles SERVERS and CLIENTS are defined in the environment. * Add the ability to tune the amount of time to pause before another retry attempt.

* Restraint's retry pause time reduced to 30 from 60. * Improve log entries to provide insight to multihost sync operations.
(Contributed by Carol Bouchard)

- FIXED: RHBZ#1756515: FALSESTRINGS not provide consistent results. If a dmesg log contains "falsestring failurestring", then falsestring will override failurestring. If they were swapped where "failurestring falsestring", then falsestring does not override failurestring which is a bug. This changeset resolves this inconsistency. It also removed printing of surrounding 5 lines around the matching line. This will make it easier for users to identify which line has matched. The full dmesg log file is also provided so user can easily search through the full dmesg log if they need to see surrounding lines.
(Contributed by Carol Bouchard)

1.10.9 Restraint 0.1.41

- FIXED: RHBZ#1753336: The cli rstrnt-adjust-watchdog command. was producing random results. The message from restraintd to the lab controller was getting truncated when the number of digits for time increased. There is an extra 30 minutes added to this message for external watchdog so it is possible for it to increase by 1 byte. Since restraintd used the same message received for the request, the message length was already set so the soup library didn't try to recalculate it. The solution is to initialize the length to 0 to force the soup library to recalculate it.
(Contributed by Carol Bouchard)
- FIXED: RHBZ#1751074: Rlse 0.1.40 seeing a lot of invalid. dmesg failures. This behavior only occurs on x86_64 arch. The rpm task /distribution/install, method VirtWorkaround() is setting an empty /usr/share/rhts/failurestrings file. As a result, every line is treated as a failure. Solution is to make sure the failurestrings file has content before using it. Included in this changeset is detail output for next triage. This output is written to the bottom of resultoutputfile.log when 01_dmesg_check reports failure. This debug code reports which set of failure and falsestring data was used: environment vars, files, or hardcoded defaults. It shows content of the failure/falsestrings variables and if the files exist, if there is data in them or the files content is also dumped into the bottom of the log file.
(Contributed by Carol Bouchard)

1.10.10 Restraint 0.1.40

Released 4 September 2019.

- FIXED: RHBZ#1609330: Restraint should have a log similar to beah's /mnt/testarea/current.log. This file points to unique task file named /tmp/tmp.XXXX (where XXXX is random). As tasks change, the link changes to new tmp.XXXX file. File current.log makes it convenient to find current task log file as the job is running.
(Contributed by Carol Bouchard)
- NEW: RHBZ#1713313: Provide an option for not rebooting the test box after localwatchdog killed a task. No new code was written for this since an option already existed. This changeset documents the option *RSTRNT_DISABLED* which allows the user to disable specified plugins.
(Contributed by Carol Bouchard)
- FIXED: RHBZ#1678549: Restraint starts too early for the system to get ready for testing. Instead, wait until network is up before starting restraint.
(Contributed by Martin Styk)
- FIXED: RHBZ#1694221: SELinux tests break. The *20_unconfined* plugin currently checks if process running with SELinux role and domain but was missing check if user is SELinux user.
(Contributed by Martin Styk)

- FIXED: RHBZ#1478653: [RESTRAINT] Error uploading /var/log/messages. Seeing error Bad Request [soup_http_error_quark, 400]. This error occurs because restraint reports the number of bytes to send but then sends more as the file continues to grow. Now we only send the number of bytes from the point the transmission began and ignore subsequent lines in the log as they are just extra noise.
  (Contributed by Carol Bouchard)
- FIXED: RHBZ#1700886: Restraint not uploading resultoutputfile.log when local watchdog expires. The variable OUTPUTFILE was not being set. It is now set to the tasks current.log (ref: 1609330) so it is now reported.
  (Contributed by Carol Bouchard)
- FIXED: RHBZ#1730617: Multihost: Task execution synchronization does not work in restraint. As documented in Beaker's Multihost Tasks section, Task 1 on both server and client must complete before moving on to Task 2 and so on. A new plugin *85_sync_multihost_tasks* was added to cause synchronization between client and server tasks.
  (Contributed by Carol Bouchard)
- FIXED: RHBZ#1700915: Resolve inconsistency of MAXTIME vs MAX_TIME variables. To resolve confusion, *RSTRNT_MAX_TIME* is being deprecated with an existing variable *KILLTIMEOVERRIDE*. This changeset documents this deprecation.
  (Contributed by Tomas Klohna)
- NEW: RHBZ#1700926: Allow task to adjust local watchdog. The command rstrnt-adjust-watchdog only affects the external watchdog. To be compatible with beah, this commmand also works for the local watchdog.
  (Contributed by Carol Bouchard)
- FIXED: RHBZ#1705223: Incomplete doc in regards to metadata/testinfo.desc. This is a spinoff from BZ1120496 but for restraint. This changeset identified and documented variables in metadata and testinfo file.
  (Contributed by Carol Bouchard)

1.10.11 Restraint 0.1.39

Released 27 February 2019.

- NEW: RHBZ#1552199: Restraint-client now supports changing timeout value for the request.
  (Contributed by Martin Styk)
- FIXED: RHBZ#1670377: Fixed compilation issues for GCC9/Automake.
  (Contributed by Martin Styk)

1.10.12 Restraint 0.1.38

Released 29 January 2019.

- FIXED: RHBZ#1670111: Fixed crash of Restraint for ppc64le and aarch64 architecture.
  (Contributed by Bill Peck)

1.10.13 Restraint 0.1.37

Released 11 January 2019.

- NEW: RHBZ#1665390: Added feature to set family from client XML.
  (Contributed by Bill Peck)
- NEW: RHBZ#1656466: Restraint now supports `@module` syntax for dependencies for RHEL8+.

(Contributed by Martin Styk)

- FIXED: RHBZ#1663125: Restraint now listens separately for IPv4 and IPv6. One running version of the protocol is sufficient for `restraintd` run.
  (Contributed by Bill Peck)
- FIXED: RHBZ#1663825: When BootCurrent is not available, Restraint will try to fall back to `/root/EFI_BOOT_ENTRY.TXT`.
  (Contributed by Martin Styk)
- FIXED: RHBZ#1659353: Fixed obsolete URL for Bzip2 package in Makefile.
  (Contributed by Martin Styk)
- FIXED: RHBZ#1599550: Fixed crash of Restraint for RHEL6 arch s390 caused by glib2.
  (Contributed by Matt Tyson)
- FIXED: RHBZ#1608262: Fixed guest-host synchronization.
  (Contributed by Dan Callaghan)

1.10.14 Restraint 0.1.36

Released 24 August 2018.

- NEW: RHBZ#1506064: The dmesg error checking plugin can now match patterns against multi-line "cut here" style traces. The plugin now ignores a warning about "mapping multiple BARs" on IBM x3250m4 systems, matching the existing behaviour of the RHTS dmesg checker.
  (Contributed by Jacob McKenzie)
- FIXED: RHBZ#1592376: Restraint resets the SIGPIPE handler before executing task processes. Previously the tasks would inherit the "ignore" action for SIGPIPE from the Restraint parent process, which would prevent normal shell broken pipe handling from working correctly in the task.
  (Contributed by Matt Tyson)
- FIXED: RHBZ#1595167: When the local watchdog timer expires, Restraint will now upload the output from **`journalctl`** in favour of `/var/log/messages` if the systemd journal is present. Previously it would attempt to upload `/var/log/messages` even if the file did not exist, causing the local watchdog handling to enter an infinite loop.
  (Contributed by Matt Tyson)
- FIXED: RHBZ#1593595: Fixed an improper buffer allocation which could cause Restraint to crash with a segmentation fault instead of reporting an error message in certain circumstances.
  (Contributed by Róman Joost)
- FIXED: RHBZ#1600825: Fixed a file conflict introduced in Restraint 0.1.35 between the `restraint` package and the `rhts-test-env` package.
  (Contributed by Matt Tyson)
- FIXED: RHBZ#1601705: Fixed a shell syntax error in the RPM %post scriptlet on RHEL4 which caused the package to be un-installable.
  (Contributed by Dan Callaghan)
- FIXED: RHBZ#1585904: Fixed a shell syntax error in the restraintd init script which caused it to fail to start on RHEL4.
  (Contributed by Dan Callaghan)

1.11 Developer Guide

If you have questions related to Restraint's development that are not currently answered in this guide, the two main ways to contact the Restraint development team are the same as those for getting general assistance with using and installing Beaker:

- the development mailing list
- the #beaker IRC channel on FreeNode

This document focuses on the mechanics of working with Restraint's code base with the target audience being a Restraint user interested in learning more about Restraint's working or a potential Restraint contributor.

1.11.1 Getting Started

Restraint is written in C. The source lives in a git repo on http://github.com/beaker-project/ along with other related projects. The following creates a local clone of the Restraint source.

```
git clone git@github.com:beaker-project/restraint.git
```

Restraint uses a number of external libraries/tools, so before you can build Restraint you need to install the external libraries using `dnf builddep restraint.spec`. Once you have installed these dependencies, running a `make all` at the source directory root will compile and build restraint, restraintd, and commands. To also run a quick sanity check, it is a good idea to run the unit tests using `make check`. The unit tests use a simple Python HTTP server and `git-daemon`, so you will need to install this as well (`dnf install git-daemon`).

1.11.2 Testing Changes

If you have fixed an existing bug or implemented a new feature, it is a good idea to add a relevant test. The existing tests can be found in the `src/` directory in the source files with names starting with `test_`.

It may also be a good idea to run a recipe by building the Restraint daemon and client from the modified code base. You can build the binaries using `make all` in the `src` directory.

From the same directory, run the restraint client with a reference to a job.xml. The following shows how to initiate the restraint client to execute a recipe:

```
restraint --host 1=127.0.0.1 --job /path/to/job.xml --restraint-path /my_development_
↪path/restraint/src/restraintd
```

Developers should use the option `--restraint-path` to point to the development path of the restraintd server. More details on this can be found in *Running Standalone*.

1.11.3 Submitting a Patch

All patches are submitted using GitHub Pull Requests feature. To do that you have to have the fork of the Restraint.

Created patch should also have a note for release notes. The note has to be created by Reno.

1.12 Guide to removing RHTS from Jobs

For some products, Test Requirements include running restraint by itself. This requires the exclusion of the legacy RHTS package or *restraint-rhts* package installation. Below lists areas to draw attention in order to eliminate RHTS

references.

1. Use the *restraint* harness package and not *restraint-rhts* in your jobs.
2. Avoid defining tasks or dependencies which cause installation of the *RHTS* library.
3. Replace *RHTS* scripts with *Restraint* scripts. *Replacement for RHTS Scripts* provides a table which maps legacy to restraint scripts.
4. Change your tasks to utilize *Restraint's metadata* file instead of *RHTS testinfo.desc* file. *Replacement for RHTS testinfo.desc File* provides details on mapping legacy *testinfo.desc* variables to restraint *metadata* variables. Depending on how your task is written, you may have to update or remove *Makefiles* so they do not process the *testinfo.desc* file. An example of this is also included in the referenced section.
5. Replace *RHTS* environment variables with *Restraint* variables. A table listing *RHTS Legacy Variables* to *Restraint Substitute* can be found in *Legacy RHTS Task Environment Variables*.

1.12.1 Replacement for RHTS Scripts

The table below lists known legacy RHTS commands. Some are provided in the *restraint-rhts* package and some are from *rhts* package. It is encouraged for people to use *Restraint*'s substitute for these commands as they are actively supported. Included in the table are the *Restraint* substitutes and which RHTS commands are deprecated.

RHTS Legacy Script	Restraint Substitute
rhts-abort	rstrnt-abort
rhts-backup	rstrnt-backup
rhts-db-submit-result, rhts_db_submit_result	rstrnt-report-result.d plugin *Report Result* (See Note)
rhts-environment.sh, rhts_environment.sh	None
rhts-extend	rstrnt-adjust-watchdog
rhts-flush	None
rhts-lint	None
rhts-power	None
rhts-reboot	rstrnt-reboot
rhts-recipe-sync-block, rhts_recipe_sync_block	rstrnt-sync-block
rhts-recipe-sync-set, rhts_recipe_sync_set	rstrnt-sync-set
rhts-report-result	rstrnt-report-result
rhts-restore	rstrnt-restore
rhts-run-simple-test	None
rhts-submit-log, rhts_submit_log	rstrnt-report-log
rhts-sync-block, rhts_sync_block	rstrnt-sync-block
rhts-sync-set, rhts_sync_set	rstrnt-sync-set
rhts-system-info	localwatchdog.d 20_sysinfo plugin *Local Watchdog* (See Note)

Note: Some functionality in *RHTS* scripts are replaced by *Restraint* plugins. Links for details on those plugins are contained in the *Restraint Substitute* column.

1.12.2 Replacement for RHTS testinfo.desc File

Legacy *RHTS* tests use the *testinfo.desc* file for their metadata[1]. *Restraint* supports generating (via the Makefile) and reading this file; however, *Restraint* does not process all the fields in this file. *Restraint* gives the *metadata* file

[1] RHTS Task Metadata.

predecence over the *testinfo.desc* file.

The tables below shows is a list of *testinfo.desc* variables *Restraint* parses and acts on. The table also shows the mapping to *Restraint* metadata section/variable.

testinfo.desc variable	metadata [section] variable substitute
Name	[General] name
Environment	[restraint] environment
TestTime	[restraint] max_time
Requires	[restraint] dependencies
RhtsRequires	[restraint] dependencies
USE_PTY	[restraint] use_pty

The following are informational variables and should be maintained. *Restraint* does not perform any action on these variables.

testinfo.desc variable	metadata [section] variable substitute
License	[General] license
Owner	[General] owner
Description	[General] description
Confidential	[General] confidential
Destructive	[General] destructive

There are no substitutes for the following *Makefile/testinfo.desc* variables in *Restraint's* metadata file. Some of these variables are informational and can be added in the metadata file but it is just documentation. *Restraint* will not act on them and they will be ignored.

- TESTVERSION
- FILES
- BUILT_FILES
- TEST_DIR
- Path
- Architectures
- Bugs
- Priority
- Releases
- RhtsOptions
- TestVersion

Example of removing testinfo.desc file

The sample files below show converting *Makefile/testinfo.desc* to *metadata* file. The Makefile does not have to be removed in its entirety. In the Sample Makefile, everything from *rhts-make.include* below should be removed. If the upper part of the Makefile is kept, the *entry_point* variable defined in the *metadata* file is not required since *Restraint* will perform *make run* when *entry_point* is not present.

Sample Makefile:

```
export TEST=/examples/no-rhts/sample-before
export TESTVERSION=1.0

BUILT_FILES=

FILES=$(METADATA) runtest.sh Makefile PURPOSE

.PHONY: all install download clean

run: $(FILES) build
        ./runtest.sh

build: $(BUILT_FILES)
        test -x runtest.sh || chmod a+x runtest.sh

clean:
        rm -f *~ $(BUILT_FILES)

include /usr/share/rhts/lib/rhts-make.include

$(METADATA): Makefile
        @echo "Owner:           User ABC1 <userabc1@example.com>" > $(METADATA)
        @echo "Name:            $(TEST)" >> $(METADATA)
        @echo "TestVersion:     $(TESTVERSION)" >> $(METADATA)
        @echo "Path:            $(TEST_DIR)" >> $(METADATA)
        @echo "Description:     Sample-before-no-rhts" >> $(METADATA)
        @echo "Type:            Sanity" >> $(METADATA)
        @echo "TestTime:        5m" >> $(METADATA)
        @echo "Priority:        Normal" >> $(METADATA)
        @echo "License:         GPLv2+" >> $(METADATA)
        @echo "Confidential:    no" >> $(METADATA)
        @echo "Destructive:     no" >> $(METADATA)
        @echo "Releases:        -RHEL7 -RHEL8" >> $(METADATA)
        @echo "Architectures:   x86_64" >> $(METADATA)
```

Makefile generated *testinfo.desc* file:

```
Owner:           User ABC1 <userabc1@example.com>
Name:            /examples/no-rhts/sample-before
TestVersion:     1.0
Path:            /mnt/tests/examples/no-rhts/sample-before
Description:     Sample-before-no-rhts
Type:            Sanity
TestTime:        5m
Priority:        Normal
License:         GPLv2+
Confidential:    no
Destructive:     no
Releases:        -RHEL7 -RHEL8
Architectures:   x86_64
```

Replacement restraint *metadata* file with no Makefile:

```
[General]
description=Sample-after-no-rhts
owner=User ABC1 <userabc1@example.com>
license=GPLv2+
```

(continues on next page)

(continued from previous page)

```
confidential=no
destructive=no

[restraint]
entry_point=./runtest.sh
max_time=5m
name=/examples/no-rhts/sample-after
```

1.12.3 Legacy RHTS Task Environment Variables

When the *testinfo.desc* file is present, *Restraint* exports the *RHTS* Legacy variables to support legacy tests written for *RHTS* (Red Hat Test System). Both the *testinfo.desc* file and these variables are being deprecated and the table below lists the variable substitutes.

RHTS Legacy Variable	Restraint Substitute
ARCH	RSTRNT_OSARCH
DISTRO	RSTRNT_OSDISTRO
FAMILY	RSTRNT_OSMAJOR
JOBID	RSTRNT_JOBID
REBOOTCOUNT	RSTRNT_REBOOTCOUNT
RECIPESETID	RSTRNT_RECIPESETID
RECIPEID	RSTRNT_RECIPEID
RECIPETESTID	RSTRNT_RECIPEID
RESULT_SERVER	No equivalent. Communication only with client/lab controller.
SUBMITTER	RSTRNT_OWNER
TASKID	RSTRNT_TASKID
TESTID	RSTRNT_TASKID
TESTNAME	RSTRNT_TASKNAME
TESTPATH	RSTRNT_TASKPATH
VARIANT	RSTRNT_OSVARIANT

CHAPTER 2

Additional Information

CHAPTER 3

Indices and Tables

- genindex
- modindex
- search

Index

Symbols

C

E

L

M

N

T