The Console Service

Table of contents

1 Overview

The console service provides an interface for automating The Grinder. It allows The Grinder to be controlled by a scheduler or a Continuous Integration framework such as Hudson/Jenkins; remote monitoring using a web browser; and creative possibilities such monitoring and influencing the test execution from a test script, perhaps by starting additional worker processes.

You can use the console service to start and stop worker processes; change console options; distribute script files; start and stop recordings; and obtain aggregated test results.

The first version of the console service was released as part of The Grinder 3.10, and provides REST web services. Future releases will provide other flavours of interface, such as a browser-based user interface, and event-driven publication of data.

2 Configuration

The console hosts an HTTP server that runs the console service. When the console is started, the server listens for HTTP requests on port 6373. For most users, the console service should work out of the box with no further configuration.

If port 6373 is unavailable, an error message will be presented. This usually occurs because another program has claimed the port. Perhaps there two copies of the console have been started. You can change the HTTP port using the console options, and also set the HTTP host to your publicly accessible host name or IP address. In fact, unless you change the host name, the HTTP server will listen on localhost, and you'll only be able to connect to the console from local processes.

You can check that the console service has started correctly by using your browser to access http://localhost:6373/version. If the service is running, the browser will display the version of The Grinder.

2.1 Running without a GUI

If you don't use the graphical user interface (../g3/console.html) , you can start the console in in a terminal mode by passing a `-headless` option as follows.

```
java -classpath lib/grinder.jar net.grinder.Console -headless
```

2.2 Setting the HTTP address and port on the command line

You can also specify the console service address and port on the command line, overriding the console options:

```
java -classpath lib/grinder.jar -Dgrinder.console.httpHost=myhost -
Dgrinder.console.httpPort=8080 net.grinder.Console
```

Here `myhost` should resolve to a local IP address.

3 The REST interface

The REST interface accepts HTTP `GET`, `POST`, and `PUT` requests. The request's `Accept` header is used to select the formatting of the response.

Accept header	Response body format
`application/clojure`	Clojure data structure
`application/json`	JSON
`application/x-yaml`	YAML
`text/html`	YAML wrapped in HTML
No accept header	JSON
Other values	*406 Not Acceptable*

The YAML in HTML support allows simple access to some of the services (those that use `GET`) from a web browser.

Some of the `POST` and `PUT` requests require additional data to be supplied in the body of the request. The request's `Content-Type` header is used to determine whether the request body should be parsed as JSON, YAML, or a Clojure data structure.

Content-Type header	Request body format
`application/clojure` `application/x-clojure`	Clojure map
`application/json` `application/x-json`	JSON object

`application/yaml`
`application/x-yaml`
`text/yaml`
`text/x-yaml`
	YAML map

Other values	*Ignored*

3.1 Available services

The following services are available.

Method	URL	Description
`POST`	`/agents/start-workers`	Send a start signal to the agents to start worker processes. Equivalent to the start processes (../g3/console.html#process-controls) button.
`GET`	`/agents/status`	Returns the status of the agent and worker processes.
`POST`	`/agents/stop`	Terminates all agents and their worker processes. You will usually want `/agents/stop-workers` instead.

Method	URL	Description
POST	/agents/stop-workers	Send a stop signal to connected worker processes. Equivalent to the reset processes (../g3/console.html#process-controls) button.
POST	/files/distribute	Start the distribution of files to agents that have an out of date cache. Distribution may take some time, so the service will return immediately and the files will be distributed in proceeds in the background. The service returns a map with an :id entry that can be used to identify the particular distribution request.
GET	/files/status	Returns whether the agent caches are stale (i.e. they are out of date with respect to the console's central copy of the files), and the status of the last file distribution.
GET	/properties	Return the current values of the console options.
PUT	/properties	Set console options. The body of the request should be a map of keys to new values; you can provide some or all of the properties. A map of the keys and their new values will be returned. You can find out the names of the keys by issuing a GET to /properties.
POST	/properties/save	Save the current console options in the preferences file. The preferences file is called .grinder_console and is stored in the home directory of the user account that is used to run the console.
GET	/recording/data	Return the current recorded data. Equivalent to the data in the results tab (../g3/console.html#Results) .
GET	/recording/data-latest	Return the latest sample of recorded data. Equivalent to the data in the lower pane of the results tab (../g3/console.html#Results) .
POST	/recording/start	Start capturing data. An initial number of samples may be

Method	URL	Description
		ignored, depending on the configured console options.
`POST`	`/recording/stop`	Stop the data capture.
`GET`	`/recording/status`	Return the current recording status.
`POST`	`/recording/reset`	Discard all recorded data. After a reset, the model loses all knowledge of Tests; this can be useful when swapping between scripts. It makes sense to reset with the worker processes stopped.
`POST`	`/recording/zero`	Reset the recorded data values to zero.
`GET`	`/version`	Returns the version of The Grinder.

4 Example session

Let's have a look at an example terminal session that exercises the REST interface. We'll use curl (http://curl.haxx.se/) as a client, but other HTTP clients will work will as well.

> **Note:**
> A web cast of a similar example session is available on YouTube (http://www.youtube.com/watch?v=OzB3bvQnS7U) .

4.1 Starting up

First, we start the console, specifying `-headless` because we're not going to be using the GUI.

```
% java -classpath lib/grinder.jar net.grinder.Console -headless

2012-05-30 18:33:30,472 INFO  console: The Grinder 3.10-SNAPSHOT
2012-05-30 18:33:30,505 INFO  org.eclipse.jetty.server.Server: jetty-7.6.1.v20120215
2012-05-30 18:33:30,538 INFO  org.eclipse.jetty.server.AbstractConnector:
 Started SelectChannelConnector@:6373
```

You can see the console service is listening on port 6373, as expected. Now open another terminal window, and check the lights are on.

```
% curl http://localhost:6373/version

The Grinder 3.10-SNAPSHOT
```

The console service has responded with the appropriate version string, as expected.

Next let's ask for the current console options.

```
% curl http://localhost:6373/properties

{"httpPort":6373,"significantFigures":3,"collectSampleCount":0,
```

```
"externalEditorCommand":"","consolePort":6372,"startWithUnsavedBuffersAsk":true,
"scanDistributionFilesPeriod":6000,"resetConsoleWithProcesses":false
"sampleInterval":3100,"resetConsoleWithProcessesAsk":true,
"frameBounds":[373,168,1068,711],"httpHost":"","externalEditorArguments":"",
"ignoreSampleCount":0,"consoleHost":"","distributeOnStartAsk":false,
"propertiesNotSetAsk":true,"distributionDirectory":"/tmp/grinder-3.9.1/foo",
"propertiesFile":"/tmp/grinder-3.9.1/foo/grinder.properties",
"distributionFileFilterExpression":
"^CVS/$|^\\.svn/$|^.*~$|^(out_|error_|data_)\\w+-\\d+\\.log\\d*$",
"saveTotalsWithResults":false,"stopProcessesAsk":true,"lookAndFeel":null}
```

The console options are returned in the response body as a JSON object containing key/value pairs. This format is easily to parse with a scripting language, or JavaScript in a browser.

4.2 Setting the properties

Some of the console options are only relevant to the GUI, but others also affect the console service. The following command changes the distribution directory to the examples directory in our distribution, and selects the `grinder.properties` file.

```
% curl -H "Content-Type: application/json" -X PUT http://localhost:6373/properties
    -d '{"distributionDirectory":"examples", "propertiesFile":"grinder.properties"}'

{"propertiesFile":"grinder.properties","distributionDirectory":"examples"}
```

The properties that were changed are returned in the response body.

4.3 Connecting an agent

In a third terminal window, let's start an agent. We'll be distributing files to the agent which it will cache in its working directory, so we'll do so in a temporary directory.

```
% cd /tmp
% java -classpath ${GRINDER_HOME}/lib/grinder.jar net.grinder.Grinder

2012-05-30 18:54:30,674 INFO  agent: The Grinder 3.10-SNAPSHOT
2012-05-30 18:54:30,737 INFO  agent: connected to console at localhost/127.0.0.1:6372
2012-05-30 18:54:30,737 INFO  agent: waiting for console signal
```

The agent has connected to the console. We could start up other agents, perhaps on other machines; we'd just need to add `-Dgrinder.console.Host=console-machine` before `net.grinder.Grinder`.

We can confirm that the console knows about the agent.

```
% curl http://localhost:6373/agents/status

[{"id":"paston02:968414967|1338400470671|425013298:0","name":"paston02","number":-1,
"state":"RUNNING","workers":[]}]
```

The agent is running, and it has not yet started any worker processes. Now we'll distribute the scripts to the agent.

```
% curl -X POST http://localhost:6373/files/distribute

{"id":1,"state":"started","files":[]}
```

File distribution is asynchronous - the result indicates that the distribution request has been queued, and allocated id 1. We can find out where it's got to by querying the status.

```
% curl http://localhost:6373/files/status

{"stale":false,"last-distribution":{"per-cent-complete":100,"id":1,"state":"finished",
"files":
["cookies.py","digestauthentication.py","ejb.py","jdbc.py","httpg2.py","console.py",
"slowClient.py","httpunit.py","sequence.py","jmssender.py","grinder.properties","sync.py",
"amazon.py","helloworldfunctions.py","form.py","xml-rpc.py","parallel.py","jaxrpc.py",
"scenario.py","threadrampup.py","statistics.py","jmsreceiver.py","helloworld.py",
"helloworld.clj","proportion.py","fba.py","scriptlifecycle.py","email.py","http.py"]}}
```

This tells us that the agent caches are no longer stale, and the distribution 1 completed, sending the list of files to the agents.

4.4 Starting the workers

We're going to have The Grinder start some worker processes and run the helloworld.py (../g3/script-gallery.html#helloworld.py) script, which is one of the files we've just sent.

We previously set the console option *propertiesFile* to a properties file in the distributed files (we chose `grinder.properties`). Setting this option causes the agent to first look for any script file in its distribution cache, falling back to its working directory if the file isn't found. We can override the values in the distributed `grinder.properties` file in properties sent with the start command.

> **Note:**
> Distributing the files to the agents is optional. If you do so, then be sure to set *propertiesFile* to a valid properties file in the distribution. Otherwise, the agent will resolve the script file name relative to its working directory, ignoring the files in the distribution cache. If you don't distribute the files you'll have to make sure the agent can find the script through some other means, such as a file system share.
> Properties supplied with the start command override those specified with *propertiesFile*, which in turn override those specified as system properties on the agent or worker process command lines, which in turn override those found in a `grinder.properties` file in the agent's working directory.

The following starts two worker processes, to perform three runs of *helloworld.py*, using five worker threads each.

```
% curl -H "Content-Type: application/json" -X POST http://localhost:6373/agents/start-
workers -d '{"grinder.processes" : "2", "grinder.threads" : "5", "grinder.runs" : "3",
 "grinder.script" : "helloworld.py" }'

success
```

4.5 Obtaining the results

Let's stop the recording. Until we do this, the TPS will be calculated over an increasing duration, and steadily fall. When doing real tests, it's more common to set `grinder.runs` to `0` so that the workers don't stop until instructed to do so, and to record a period of data before they are stopped.

```
% curl -X POST http://localhost:6373/recording/stop

{"state":"Stopped","description":"Collection stopped"}
```

We can now retrieve the recording data.

```
% curl http://localhost:6373/recording/data

{"status":{"state":"Stopped","description":"Collection stopped"},
"columns":["Tests","Errors","Mean Test Time (ms)","Test Time Standard Deviation
 (ms)","TPS","Peak TPS"],
"tests":[{"test":1,"description":"Log method","statistics":
[30,0,0.2,0.4,9.674298613350532,
9.67741935483871]}],
"totals":[30,0,0.2,0.4,9.674298613350532,9.67741935483871]}
```

There were 30 executions of Test 1 as expected (2 worker processes x 5 worker threads x 3 runs), with an average execution time of 0.2 ms.

```
% curl http://localhost:6373/recording/data-latest

{"status":{"state":"Stopped","description":"Collection stopped"},
"columns":["Tests","Errors","Mean Test Time (ms)","Test Time Standard Deviation
 (ms)","TPS","Peak TPS"],
"tests":[{"test":1,"description":"Log method","statistics":
[30,0,0.2,0.4,9.674298613350532,
9.67741935483871]}],
"totals":[30,0,0.2,0.4,9.674298613350532,9.67741935483871]}
```

Adding the -latest will retrieve the latest sample data available. This is most useful to get near real time data a currently executing test.
Again, there were 30 executions of Test 1 as expected (2 worker processes x 5 worker threads x 3 runs), with an average execution time of 0.2 ms.

4.6 Conclusion

I hope you've enjoyed this quick tour of the console service. Start the console and an agent yourself, and have a play.

> Note:
>
> **Tips**
> If a call to a service results in *Resource not found*, check you've used the appropriate HTTP method (GET, PUT, or POST).
> You might find it simpler to run the console GUI (don't add `-headless` to the command line). This will allow you to see the current console status at a glance.