Megatest User Manual

Matthew Welland

Document Number: TSETAGEM-011
Revision Number: v1.36
Jan. 29, 2012

Email: matt@kiatoa.com.

Web: www.kiatoa.com/fossils/megatest

Megatest/document Revision History

Notable revisions of the software are occasionally documented here.

Version	Author	Description	Date
v1.25	matt	converted to new document template	Jan. 29, 2012

Contents

1 Introduction

1.1 Megatest design philosophy

Megatest is intended to provide the minimum needed resources to make writing a suite of tests and implementing continuous build for software, design engineering or process control (via owlfs for example) without being specialized for any specific problem space. Megatest in of itself does not know what constitutes a PASS or FAIL of a test. In most cases megatest is best used in conjunction with logpro or a similar tool to parse, analyze and decide on the test outcome.

1.2 Megatest architecture

All data to specify the tests and configure the system is stored in plain text files. All system state is stored in an sqlite3 database. Tests are launched using the launching system available for the distributed compute platform in use. A template script is provided which can launch jobs on local and remote Linux hosts. Currently megatest uses the network filesystem to "call home" to your master sqlite3 database.

2 Installation

2.1 Dependencies

Chicken scheme and a number of "eggs" are required for building megatest. See the file utils/installall.sh for an automated way to install the dependencies on Linux.

2.2 Build and Install

Run "make test" to create the megatest executable. You may wish to copy the executable to a centrally accessible location.

3 Setup

3.1 Create megatest.config

Create the file megatest.config using the megatest.config template from the tests directory. At a minimum you need the following:

```
# Fields are the keys under which your test runs are organized
[fields]
field1 TEXT
field2 TEXT
```

```
[jobtools]
# The launcher launches jobs to the local or remote hosts,
# the job is managed on the target host by megatest,
# comment out launcher to run local only. An example launcher
# "nbfake" can be found in the utils directory.
launcher nbfake

# The disks section specifies where the tests will be run. As you
# run out of space in a partition you can add additional disks
# entries.
# Format is:
# name /path/to/area
[disks]
disk1 /tmp
```

3.2 Create runconfigs.config

This file is used to set environment variables that are run specific. You can simply create an empty file to start.

```
# runconfigs.config
```

3.3 Create the tests directory and your first test

```
mt
|-- megatest.config
|-- runconfigs.config
`-- tests
    `-- mytest
        |-- main.sh
        `-- testconfig
```

3.4 Create the testconfig file for your test

```
[setup]
runscript main.sh
```

3.5 Create your test running script, main.sh

```
#!/bin/bash

$MT_MEGATEST -runstep mystep1 "sleep␣20;echo␣Done" -m "mystep1␣is␣done"
$MT_MEGATEST -test-status :state COMPLETED :status PASS -m "This␣is␣a␣comment"
```

3.6 Run megatest and watch your run progress

```
megatest :field1 abc :field2 def :runname 2011week08.4a -runall

watch megatest -list-runs %

# OR use the dashboard

dashboard &
```

4 Choose Flow or Unstructured Run?

A flow is a structured and specifically sequenced set of tests. See the Flows chapter to understand the difference.

5 How to Write Tests

5.1 A Simple Test with one Step

```
mkdir simpletest
cd simpletest
```

5.2 Create your testconfig file

```
# testconfig

[setup]
runscript main.csh
```

5.3 Create the main.csh script

Note: Using csh is NOT recommended. Use bash, perl, ruby, zsh or anything other than csh. We use csh here because it is popular in the EDA industry for which Megatest was originally created.

```
#!/bin/tcsh -x

# run the cpu1 simulation.
#   The step name is "run_simulation"
#   The commandline being run for this step is "runsim cpu1"
#   The logpro file to validate the output from the run is "runsim.logpro"

$MT_MEGATEST -runstep run_simulation -logpro runsim.logpro "runsim␣cpu1"
$MT_MEGATEST -test-status :state COMPLETED :status $?
```

You can now run megatest and the created test directory will contain the new files “run_simulation.html” and “run_simulation.log”. If you are using the dashboard you can click on the run and then push the “View log” button to view the log file in firefox.

5.4 Simple Test with Multiple Steps

To run multiple steps simply add them to the main.csh file. Here we add a step to test “cpu2”. The second step that tests cpu2 will only run after the step that tested “cpu1” completes.

```
#!/bin/tcsh -x

# run the cpu1 simulation.
#   The step name is "run_simulation"
#   The commandline being run for this step is "runsim cpu1"
#   The logpro file to validate the output from the run is "runsim.logpro"

$MT_MEGATEST -runstep run_simulation_cpu1 -logpro runsim.logpro "runsim␣cpu1" && \
$MT_MEGATEST -runstep run_simulation_cpu2 -logpro runsim.logpro "runsim␣cpu2"
$MT_MEGATEST -test-status :state COMPLETED :status $?
```

6 Simple Test with Multiple Steps, Some in Parallel

6.1 The Makefile

A good way to run steps in parallel within a single test, especially when there are following steps, is to use the Unix Make utility. Writing Makefiles is beyond the scope of this document but here is a minimal example that will run “runsim cpu1” and “runsim cpu2” in parallel. For more information on make try “info make” at the Linux command prompt.

```
# Example Makefile to run two steps in parallel

RTLDIR=/path/to/rtl
CPUS = cpu1 cpu2

run_simulation_$(CPUS).html : $(RTLDIR)/$(CPUS)
        $(MT_MEGATEST) -runstep run_simulation_$(CPUS) -logpro runsim.logpro "runsim␣$(
```

6.2 The main.csh file

```
#!/bin/tcsh -x

# run the cpu1 and cpu2 simulations in parallel.
# The -j parameter tells make how many jobs it may run in parallel

make -j 2
$MT_MEGATEST -test-status :state COMPLETED :status $?
```

7 Simple Test with Iteration

Since no jobs run after the cpu1 and cpu2 simulations in this test it is possible to use iterated mode.

7.1 Update your testconfig file for iteration

```
[setup]
runscript main.csh

[items]
CPU cpu1 cpu2
```

7.2 Rewrite your main.csh for iteration

```
#!/bin/tcsh -x

# run the cpu simulation but now use the environment variable $CPU
# to select what cpu to run the simulation against

$MT_MEGATEST -runstep run_simulation -logpro runsim.logpro "runsim␣$CPU"
# As of version 1.07 Megatest automatically converts a status of "0"
# to "PASS", any other number to "FAIL" and directly uses the value of
# a string passed in.
$MT_MEGATEST -test-status :state COMPLETED :status $?
```

7.3 Tests with Inter-test dependencies

Sometimes a test depends on the output from a previous test or it may not make sense to run a test is another test does not complete with status “PASS”. In either of these scenarios you can use the “waiton” keyword in your testconfig file to indicate that this test must wait on one or more tests to complete before being launched. In this example there is no point in running the “system” test if the “cpu” and “mem” tests either do not complete or complete but with status “FAIL”.

```
# testconfig for the "system" test
[setup]
runscript main.csh
waiton cpu mem
```

7.4 Rolling up Miscellaneous Data

Use the -load-test-data switch to roll up arbitrary data from a test into the test_data table.

```
# Fields are:
# category,variable,value,expected,tol,units,comment,status

$MT_MEGATEST -load-test-data << EOF
foo,bar,1.2,1.9,>
foo,rab,1.0e9,10e9,1e9
foo,bla,1.2,1.9,<
foo,bal,1.2,1.2,<,,Check for overload
foo,alb,1.2,1.2,<=,Amps,This is the high power circuit test
foo,abl,1.2,1.3,0.1
```

```
foo,bra,1.2,pass,silly stuff
faz,bar,10,8mA,,,"this␣is␣a␣comment"
EOF
```

New entries are keyed on the category and variable. If a new record is inserted with a category and variable that have already been used the new record will replace the old record.

Where value, expected and tol are specified the behavior is as follows.

- If value, expected and tol are numbers then status is calculated as PASS if (expected-tol) <= value <= (expected+tol)
- If value and expected are numbers and tol is >, <, >= or <= then value is compared with expected using the operator given by tol
- If status is specified its value overrides the above calculations.

7.5 Rolling up Runs

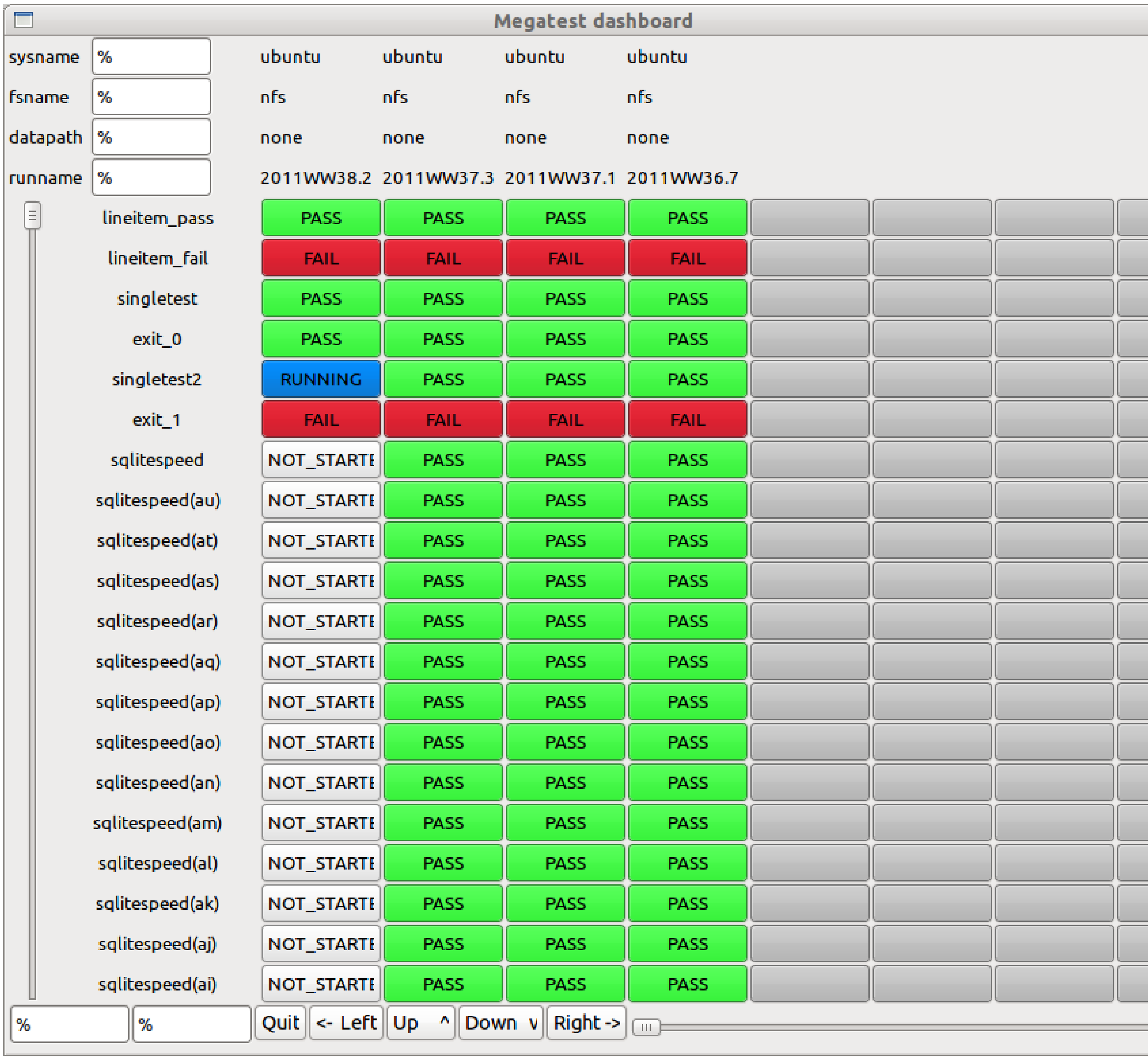
To roll up a number of tests in a sequence of runs to a single run use the -rollup command.

```
megatest -rollup :sysname ubuntu :fsname nfs :datapath none :runname rollup_ww38
```

All keys must be specified and the runname is the name of the run that will be created. All paths are kept original inside the database. When -remove-runs is used to delete runs the data is not deleted if there are rollups that refer to the data.

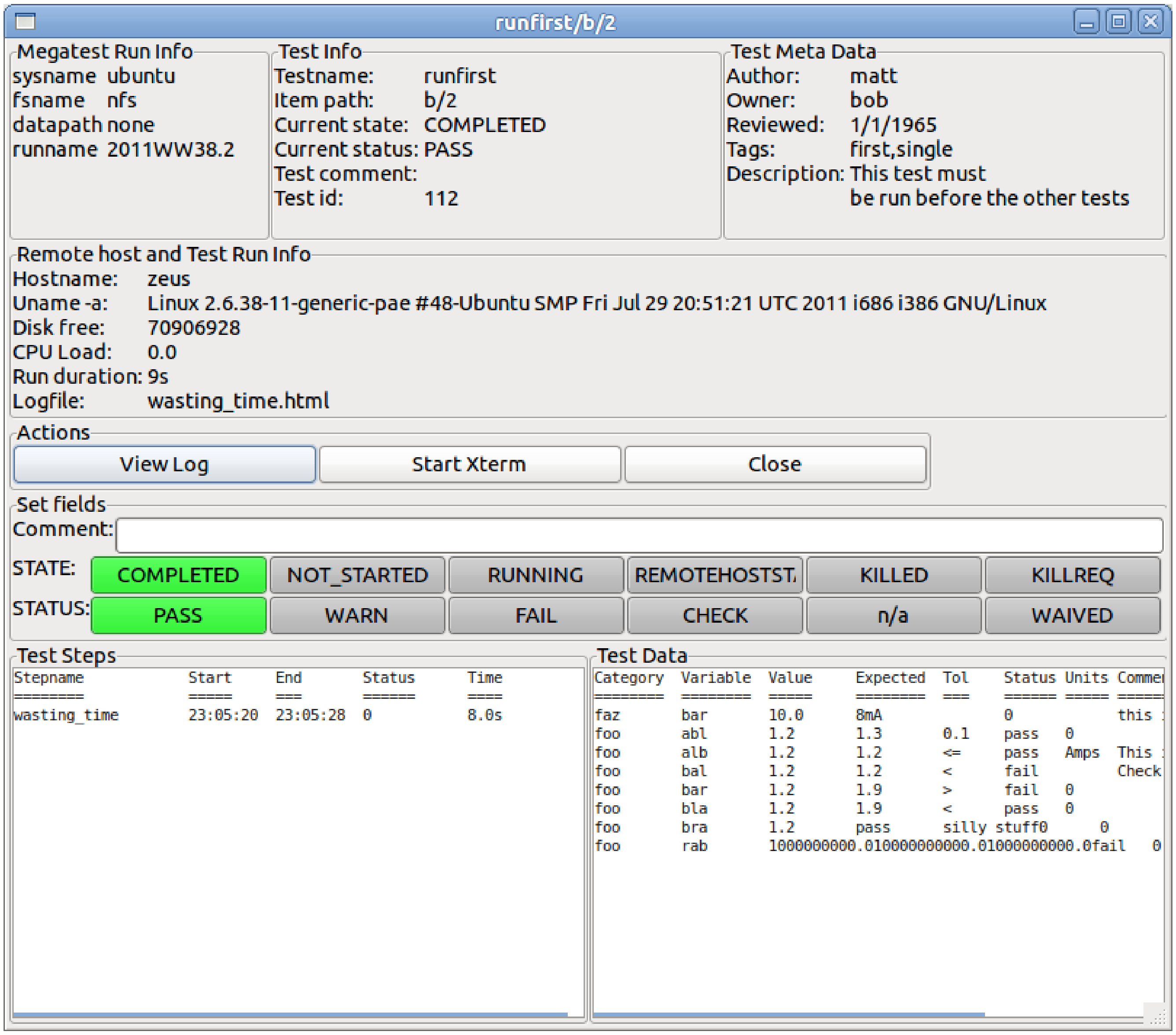
8 Dashboard

```
> dashboard &
```



sysname	%	ubuntu	ubuntu	ubuntu	ubuntu			
fsname	%	nfs	nfs	nfs	nfs			
datapath	%	none	none	none	none			
runname	%	2011WW38.2	2011WW37.3	2011WW37.1	2011WW36.7			
	lineitem_pass	PASS	PASS	PASS	PASS			
	lineitem_fail	FAIL	FAIL	FAIL	FAIL			
	singletest	PASS	PASS	PASS	PASS			
	exit_0	PASS	PASS	PASS	PASS			
	singletest2	RUNNING	PASS	PASS	PASS			
	exit_1	FAIL	FAIL	FAIL	FAIL			
	sqlitespeed	NOT_STARTE	PASS	PASS	PASS			
	sqlitespeed(au)	NOT_STARTE	PASS	PASS	PASS			
	sqlitespeed(at)	NOT_STARTE	PASS	PASS	PASS			
	sqlitespeed(as)	NOT_STARTE	PASS	PASS	PASS			
	sqlitespeed(ar)	NOT_STARTE	PASS	PASS	PASS			
	sqlitespeed(aq)	NOT_STARTE	PASS	PASS	PASS			
	sqlitespeed(ap)	NOT_STARTE	PASS	PASS	PASS			
	sqlitespeed(ao)	NOT_STARTE	PASS	PASS	PASS			
	sqlitespeed(an)	NOT_STARTE	PASS	PASS	PASS			
	sqlitespeed(am)	NOT_STARTE	PASS	PASS	PASS			
	sqlitespeed(al)	NOT_STARTE	PASS	PASS	PASS			
	sqlitespeed(ak)	NOT_STARTE	PASS	PASS	PASS			
	sqlitespeed(aj)	NOT_STARTE	PASS	PASS	PASS			
	sqlitespeed(ai)	NOT_STARTE	PASS	PASS	PASS			
%	%							

Pushing one of the buttons on the main dashboard will bring up the test specific dashboard. Values are updated in semi-real time as the test runs.


runfirst/b/2

Megatest Run Info
sysname ubuntu
fsname nfs
datapath none
runname 2011WW38.2

Test Info

Testname:	runfirst
Item path:	b/2
Current state:	COMPLETED
Current status:	PASS
Test comment:	
Test id:	112

Test Meta Data

Author:	matt
Owner:	bob
Reviewed:	1/1/1965
Tags:	first,single
Description:	This test must be run before the other tests

Remote host and Test Run Info

Hostname:	zeus
Uname -a:	Linux 2.6.38-11-generic-pae #48-Ubuntu SMP Fri Jul 29 20:51:21 UTC 2011 i686 i386 GNU/Linux
Disk free:	70906928
CPU Load:	0.0
Run duration:	9s
Logfile:	wasting_time.html

Test Steps
```
Stepname            Start     End       Status     Time
========            =====     ===       ======     ====
wasting_time        23:05:20  23:05:28  0          8.0s
```

Test Data
```
Category  Variable  Value     Expected  Tol     Status Units Commen
========  ========  =====     ========  ===     ====== ===== ======
faz       bar       10.0      8mA               0           this :
foo       abl       1.2       1.3       0.1     pass   0
foo       alb       1.2       1.2       <=      pass   Amps  This :
foo       bal       1.2       1.2       <       fail        Check
foo       bar       1.2       1.9       >       fail   0
foo       bla       1.2       1.9       <       pass   0
foo       bra       1.2       pass      silly stuff0      0
foo       rab       1000000000.010000000000.01000000000.0fail   0
```

9 Generating an OpenDocument Spreadsheet from the Database

And OpenDocument multi-paned spreadsheet can be generated from the megatest.db file by running -extract-ods

```
megatest -extract-ods results.ods :runname %
```

You can optionally specify the keys for your database to limit further the runs to extract into the spreadsheet. The first sheet contains all the run data and subsequent sheets contain data rolled up for the individual tests.

10 Introspection

10.1 Getting previous test paths

```
megatest -test-paths -target %/%/% :runname % -testpatt % -itempatt % :status PASS
```

11 Flows

A flow specifies the tests to run, the order and dependencies and is managed by a running megatest process.

12 Flow Specification and Running (Not released yet)

12.1 Write your flow file

flows/<flowname>.config

```
# Flow: <flowname>
[flowconfig]
# turn on item level dependencies
itemdeps on

[flowsteps]
# <testname>[,<predecessor>]

# Run the test "copydata"
copydata

# Run the test "setup" after copydata completes with PASS, WARN or WAIVE
setup,copydata

# once the test "setup" completes successfully run sim1, sim2 and sim3
sim1,setup
sim2,setup
sim3,setup
```

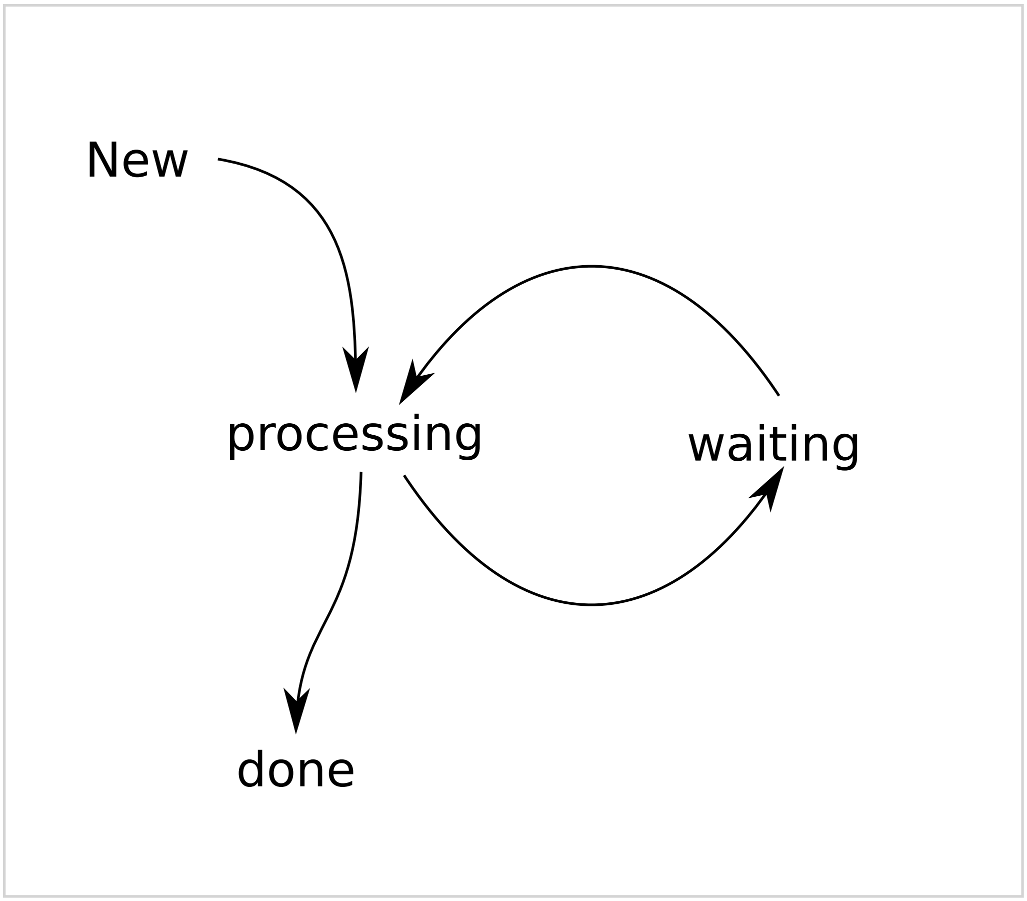
12.2 Run the flow

```
megatest -runflow <flowname> :FIELD1 val1 :FIELD2 val2 :runname wk32.4
```

13 Monitor based running

13.1 Monitor logic

Note: The monitor is usable but incomplete as of Megatest v1.31. Click on the “Monitor” button on the dashboard to start the monitor and give it a try.



14 Reference

14.1 Configuration file Syntax

Note: whitespace is preserved including at the end of line. Ensure your entries only have whitespace at the end of line when needed to avoid problems.

14.1.1 Sections

```
[section name]
```

This creates a section named “section name”

14.1.2 Variables

```
VARX has this value
```

The variable "VARX" will have the value "has this value"

14.1.3 Includes

```
[include filename]
```

The file named "filename" will be included as if part of the calling file. NOTE: This means no section can be named "include " (with the whitespace).

14.1.4 Setting a variable by running a command

```
VARNAME [system ls /tmp]
```

The variable "VARNAME" will get a value created by the Unix command "ls /tmp". All lines of output from the command will be joined with a space.

14.1.5 Notes

- Some variables are infered as lists. Each token on the line separated by whitespace will be member of the list.
- Comments (lines starting with #) and blank lines are ignored.

14.2 Environment variables

Variable	Purpose
MT_CMDINFO	Conveys test variables to the megatest test runner.
MT_TEST_RUN_DIR	Directory assigned by megatest for the test to run.
MT_TEST_NAME	Name of the test, corrosponds to the directory name under tests.
MT_ITEM_INFO	Iterated tests will set this to a sequence of key/values ((KEY val) ...)
MT_RUN_AREA_HOME	Directory where megatest was launched from and where the tests code can be found
MT_RUNNAME	Name of this run as set by the :runname parameter
MT_MEGATEST	Path/Filename to megatest executable. Found either from called path or but using the "exectuable" keyword in the [setup] section.
<field1>	The field values as set on the megatest -runall command line (e.g. :field1 abc)

14.3 megatest.config

section	variable	value	required	comment
[setup]	max_concurrent_jobs	if variable is not defined no limit on jobs	no	
	executable	full path to megatest binary	no	Use only if necessary, megatest will extract the location from where it used to launch and add append that to the PATH for test runs.
	runsdir	full path to where the link tree to all runs will be created	no	Because your runs may be spread out over several disk partitions a central link tree is created to make finding all the runs easy.
[fields]	string of letters, numbers and underscore	string of letters, numbers and underscore	at least one	
[jobtools]	launcher	command line used to launch jobs - the job command (megatest -execute) will be appended to this	no	
	workhosts	list of hostnames to run jobs on NOT SUPPORTED RIGHT NOW	n/a	
[jobgroups]	string of letters, numbers and underscore	number	no	Control number of jobs allowed to concurrently run in categories. See [jobgroup] in testconfig
[env-override]	string of letters, numbers and underscore	any string	no	These are set on the test launching machine, not the test running machine. Typical usage is to control the host or run queue for launching tests. These values will not be seen by the test when it runs.
[disks]	string of letters, numbers and underscore	a valid path writable by the test launching process and by the test process	yes	The disk usage balancing algorithm is to choose the disk with the least space for each test run.

14.4 runconfigs.config file

section	variable	value	required?	comment
[default]	string of letters, numbers and underscore	any	no	variables set in this section will be available for all runs, defining the same variable in another section will override the value from the default section
[field1value/field2value...]	string of letters, numbers and underscore	any	no	the values in this section will be set for any run where field1 is field1value, field2 is field2value and fieldN is fieldNvalue.

Example: a test suite that checks that a piece of software works correctly for different customer configurations and locations each of which is done as a separate release regression run. The fields, CUSTOMER and LOCATION were chosen. The following runconfigs.config file would set some variables specific to runs for megacorp in India and femtocorp in the Cook Islands and New Zealand:

```
# runconfigs.config
[default]
ENCRYTION true

[megacorp/india]
TESTPATH /nfs/testing/megacorp_runs

[femtocorp/cook_islands]
ENCRYTION false
TESTPATH /afs/kiatoa/testing/cook_islands

[femtocorp/new_zealand]
TESTPATH /afs/kiatoa/testing/new_zealand

[megacorp/new_zealand]
TESTPATH /nfs/testing/megacorp_runs
```

Running megatest like this: megatest :CUSTOMER megacorp :LOCATION new_zealand :runname week12_2011_run1 -runall

Would set: ENCRYPTION true

TESTPATH /nfs/testing/megacorp_runs

14.5 Writing tests

14.5.1 testconfig file

section	variable	value	required?	comments
[setup]	runscript	name of script to execute for this test	yes	The script must be executable and either provide the full path or put a copy at the top of your test directory
[requirements]	waiton	list of valid test names	no	This test will not run until the named tests are state completed and status PASS
	jobgroup			
[items]	any valid	list of values	no	The test will be repeated once for each item with the variable name set to the value. If there is more than one variable then the test will be run against all unique combinations of the values
[eztests]	any valid	stepname command	no	Use in addition to or instead of runscript for easy implementation of steps. If <stepname>.logpro exists it will be applied to the <stepname>.log and resulting exit code will be used to determine PASS/FAIL/WARN

14.5.2 Command line

switch or param	parameter	purpose	comments
-h		brief help	
-runall		run all tests	
-runtests	test1,test2,...	run one or more tests	
-step	stepname	record a step	requires :state and :status
-test-status		record the test status	requires :state and :status
-setlog	logfilename	set the logfile name for a test	path is assumed to be relati
-set-toplog	logfilename	set the logfile name for the top test in an iterated test run	each sub test can have its o
-m	“comment”	sets a comment for the step, test or run	
:runname	[a-zA-Z0-9_-]+	directory in which this run will be stored in the test run area	
:state	any value	Set the step or test state, this is stored in the state field in the steps or tests table respectively	For tests Megatest recognis
:status	any value	Set the step or test status, this is stored in the status field in the steps or tests table respectively	For tests Megatest recognis
-list-runs	any value, % is wildcard	Respects -itempatt and -testpatt for filters	
-testpatt	any value, % is wildcard		
-itempatt	any value, % is wildcard		
-showkeys		Print the keys being used for this database	
-force		Test will not re-run if in the “PASS”, “CHECK” or “KILLED”, using -force will force the run to be launched.	WARNING: The -force sw
-xterm		Launch an xterm instead of run the test. The xterm will have the environment that the test would see.	
-remove-runs		Remove a run, test or subtest from the database and the disk. Cannot be undone. Requires -testpatt, -itempatt, :runname and all keys be specified.	
Test helpers			
-runstep		Used inside a test to run a step, record the start and end of the step and optionally analyze the output using logpro.	
-logpro		If using logpro to acess the PASS/FAIL status of the step you specify the logpro file with this parameter.	

A Data

B References