Testing the RVM

Jikes RVM includes provisions to run unit tests as well as functional and performance tests. It also includes a number of actual tests, both unit and functional ones.

Unit Tests

Jikes RVM makes writing simple unit tests easy. Simply give your JUnit 4 tests a name ending in `Test` and place test sources under `rvm/test-src`. The tests will be picked up automatically.

The tests are then run on the bootstrap VM, i.e. the JVM used to build Jikes RVM. You can also configure the build to run unit tests on the newly built Jikes RVM. Note that this may significantly increase the build times of slow configurations (e.g. prototype and protype-opt).

If you are developing new unit tests, it may be helpful to run them on an existing Jikes RVM image. This can be done by using the Ant target `unit-tests-on-existing-image`. The path for the image is determined by the usual properties of the Ant build.

Functional and Performance Tests

See External Test Resources for details or downloading prerequisites for the functional tests. The tests are executed using an Ant build file and produce results that conform to the definition below. The results are aggregated and processed to produce a high level report defining the status of Jikes RVM.

The testing framework was designed to support continuous and periodical execution of tests. A "*test-run*" occurs every time the testing framework is invoked. Every "*test-run*" will execute one or more "*test-configuration*"s. A "*test-configuration*" defines a particular build "*configuration*" (See Configuring the RVM for details) combined with a set of parameters that are passed to the RVM during the execution of the tests. i.e. a particular "*test-configuration*" may pass parameters such as `-X:aos:enable_recompilation=false -X:aos:initial_compiler=opt -X:irc:O1` to test the Level 1 Opt compiler optimizations.

Every "*test-configuration*" will execute one or more "*group*"s of tests. Every "*group*" is defined by a Ant build.xml file in a separate sub-directory of `$RVM_ROOT/testing/tests`. Each "*test*" has a number of input parameters such as the classname to execute, the parameters to pass to the RVM or to the program. The "*test*" records a number of values such as execution time, exit code, result, standard output etc. and may also record a number of statistics if it is a performance test.

The project includes several different types of _test run_s and the description of each the test runs and their purpose is given in Test Run Descriptions.

> **Note**
> The buildit script provides a fast and easy way to build and the system. The script is simply a wrapper around the mechanisms described below.

Ant Properties

There is a number of ant properties that control the test process. Besides the properties that are already defined in Building the RVM the following properties may also be specified.

Property	Description	Default
test-run.name	The name of the *test-run*. The name should match one of the files located in the build/test-runs/ directory minus the '.properties' extension.	pre-commit
results.dir	The directory where Ant stores the results of the test run.	`${jikesrvm.dir}/results`
results.archive	The directory where Ant gzips and archives a copy of test run results and reports.	`${results.dir}/archive`
send.reports	Define this property to send reports via email.	(Undefined)
mail.from	The from address used when emailing report.	jikesrvm-core@lists.sourceforge.net
mail.to	The to address used when emailing report.	jikesrvm-regression@lists.sourceforge.net
mail.host	The host to connect to when sending mail.	localhost
mail.port	The port to connect to when sending mail.	25

<configuration>.built	If set to true, the test process will skip the build step for specified configurations. For the test process to work the build must already be present.	(Undefined)
skip.build	If defined the test process will skip the build step for all configurations and the javadoc generation step. For the test process to work the build must already be present.	(Undefined)
skip.javadoc	If defined the test process will skip the javadoc generation step.	(Undefined)

Defining a test-run

A *test-run* is defined by a number of properties located in a property file located in the build/test-runs/ directory.

The property *test.configs* is a whitespace separated list of *test-configuration* "tags". Every tag uniquely identifies a particular *test-configuration*. Every *test-configuration* is defined by a number of properties in the property file that are prefixed with *test.config.<tag>.* and the following table defines the possible properties.

Property	Description	Default
tests	The names of the test groups to execute.	None
name	The unique identifier for *test-configuration*.	""
configuration	The name of the RVM build configuration to test.	<tag>
target	The name of the RVM build target. This can be used to trigger compilation of a profiled image	"main"
mode	The test mode. May modify the way test groups execute. See individual groups for details.	""
extra.rvm.args	Extra arguments that are passed to the RVM. These may be varied for different runs using the same image.	""

Note
The order of the test-configurations in *test.configs* is the order that the test-configurations are tested. The order of the groups in *test.config.<tag>.test* is the order that the tests are executed.

The simplest *test-run* is defined in the following figure. It will use the build configuration "*prototype*" and execute tests in the "*basic*" group.

build/test-runs/simple.properties

```
test.configs=prototype
test.config.prototype.tests=basic
```

The test process also expands properties in the property file so it is possible to define a set of tests once but use them in multiple test-configurations as occurs in the following figure. The groups basic, optests and dacapo are executed in both the prototype and prototype-opt test\configurations.

build/test-runs/property-expansion.properties

```
test.set=basic optests dacapo
test.configs=prototype prototype-opt
test.config.prototype.tests=${test.set}
test.config.prototype-opt.tests=${test.set}
```

Test Specific Parameters

Each test can have additional parameters specified that will be used by the test infrastructure when starting the Jikes RVM instance to execute the test. These additional parameters are described in the following table.

Parameter	Description	Default Property	Default Value

initial.heapsize	The initial size of the heap.	${test.initial.heapsize}	${config.default-heapsize.initial}
max.heapsize	The initial size of the heap.	${test.max.heapsize}	${config.default-heapsize.maximum}
max.opt.level	The maximum optimization level for the tests or an empty string to use the Jikes RVM default.	${test.max.opt.level}	""
processors	The number of processors to use for garbage collection for the test or 'all' to use all available processors.	${test.processors}	all
time.limit	The time limit for the test in seconds. After the time limit expires the Jikes RVM instance will be forcefully terminated.	${test.time.limit}	1000
class.path	The class path for the test.	${test.class.path}	
extra.args	Extra arguments that are passed to the RVM.	${test.rvm.extra.args}	""
exclude	If set to true, the test will be not be executed.		""

To determine the value of a test specific parameters, the following mechanism is used;

1. Search for one of the the following ant properties, in order.
    a. test.config.<build-configuration>.<group>.<test>.<parameter>
    b. test.config.<build-configuration>.<group>.<parameter>
    c. test.config.<build-configuration>.<parameter>
    d. test.config.<build-configuration>.<group>.<test>.<parameter>
    e. test.config.<build-configuration>.<group>.<parameter>
2. If none of the above properties are defined then use the parameter that was passed to the <rvm> macro in the ant build file.
3. If no parameter was passed to the <rvm> macro then use the default value which is stored in the "Default Property" as specified in the above table. By default the value of the "Default Property" is specified as the "Default Value" in the above table, however a particular build file may specify a different "Default Value".

Excluding tests

Sometimes it is desirable to exclude tests. The test exclusion may occur as the test is known to fail on a particular target platform, build configuration or maybe it just takes too long. To exclude a test, you must define the test specific parameter "exclude" to true either in .ant.properties or in the test-run properties file.

i.e. At the time of writing the Jikes RVM does not fully support volatile fields and as a result th test named "TestVolatile" in the "basic" group will always fail. Rather than being notified of this failure we can disable the test by adding a property such as "test.config.basic.TestVolatile.exclude=true" into test-run properties file.

Executing a test-run

The tests are executed by the Ant driver script *test.xml*. The *test-run.name* property defines the particular test-run to execute and if not set defaults to "*sanity*". The command `ant -f test.xml -Dtest-run.name=simple` executes the test-run defined in *build/test-runs/simple.properties*. When this command completes you can point your browser at `${results.dir}/tests/${test-run.name}/Report.html` to get an overview on test run or at `${results.dir}/tests/${test-run.name}/Report.xml` for an xml document describing test results.