Configuring the RVM

The build process requires a number of build time parameters that specify the features and components for a Jikes RVM build. Typically the build parameters are defined within a property file located in the build/configs directory. The following table defines the parameters for the build configuration.

Property	Description	Default
config.name	A unique name that identifies the set of build parameters.	None
config.bootimage.compiler	Parameter selects the compiler used when creating the bootimage. Must be either *opt* or *base*.	base
config.bootimage.compiler.args	Parameter specifies any extra args that are passed to the bootimage compiler.	""
config.runtime.compiler	Parameter selects the compiler used at runtime. Must be either *opt* or *base*.	base
config.include.aos	Include the adaptive system if set to true. Parameter will be ignored if config.runtime.compiler is not *opt*.	false
config.mmtk.plan	The name of the GC plan to use for the build. See MMTk for more details.	None
config.default-heapsize.initial	Parameter specifying the default initial heap size in MB.	20
config.default-heapsize.maximum	Parameter specifying the default maximum heap size in MB.	100
config.assertions	Parameter specifies the level of assertions in the code base. Must be one of *extreme*, *normal* or *none*.	normal
config.stress-gc-interval	The build will stress test the gc subsytem if set to a positive value. The value indicates the number of allocations between collections	0
config.include.perfevent	Set to true to build Jikes RVM with support for performance counters.	false
config.include.gcspy	Set to true to build Jikes RVM with GCSpy support. See Using GCSpy for more details.	false
config.include.gcspy-client	Set to true to bundle the GCSpy client with the Jikes RVM build. Parameter will be ignored if config.include.gcspy is not *true*.	false
config.include.gcspy-stub	Set to true to use the GCSpy stub rather than the real GCSpy component. Parameter will be ignored if config.include.gcspy is not *true*.	false
config.include.all-classes	Include all the Jikes RVM classes in the bootimage if set to true.	false

Jikes RVM Configurations

A typical user will use one of the existing build configurations and thus the build system only requires that the user specify the config.name property. The name should match one of the files located in the build/configs/ directory minus the '`.properties`' extension.

Logical Configurations

There are many possible Jikes RVM configurations. Therefore, we define four "logical" configurations that are most suitable for casual or novice users of the system. The four configurations are:

- **prototype:** A simple, fast to build, but low performance configuration of Jikes RVM. This configuration does not include the optimizing compiler or adaptive system. Most useful for rapid prototyping of the core virtual machine.
- **prototype-opt:** A simple, fast to build, but low performance configuration of Jikes RVM. Unlike prototype, this configuration does include the optimizing compiler and adaptive system. Most useful for rapid prototyping of the core virtual machine, adaptive system, and optimizing compiler.
- **development:** A fully functional configuration of Jikes RVM with reasonable performance that includes the adaptive system and optimizing compiler. This configuration takes longer to build than the two prototype configurations.
- **production:** The same as the development configuration, except all assertions are disabled. This is the highest performance configuration of Jikes RVM and is the one to use for benchmarking and performance analysis. Build times are similar to the development configuration.

The mapping of logical to actual configurations may vary from release to release. In particular, it is expected that the choice of garbage collector for these logical configurations may be different as MMTk evolves.

Logical configurations that are not mentioned here are not recommended for novice users of the system.

Configurations in Depth

Most standard Jikes RVM configuration files follow the following naming scheme:

[ExtremeAssertions] **(Base | Full | Fast) (`Base` | `Adaptive`)** *<garbage collector>*
where

- *ExtremeAssertions* is optional. Its presence indicates that the `config.assertions` configuration parameter is set to `extreme`. This turns on a number of expensive assertions.
- **Base | Full | Fast** determines the performance of the Jikes RVM boot image. **Base** denotes baseline compiler and enabled assertions, **Full** denotes optimizing compiler and enabled assertions, **Fast** denotes optimizing compiler and disabled assertions. Note that **Fast** is exclusive with *ExtremeAssertions* and that **Full** and **Fast** imply that adaptive system and optimizing compiler are included in the image.
- **`Base` | `Adaptive`** denotes whether or not the adaptive system and optimizing compiler are included in the build.
- the *garbage collector* is the garbage collection scheme used.

Each version of Jikes RVM provides several garbage collector choices. For a definitive list of garbage collector choices, please refer to the configurations that are shipped with your version of Jikes RVM. If you need a configuration that is not available by default, you can just define your own based on the existing ones (it's easy!).

Some garbage collector suffixes that may be available are:

- "NoGC" no garbage collection is performed.
- "SemiSpace" a copying semi-space collector.
- "MarkSweep" a mark-and-sweep (non copying) collector
- "GenCopy" a classic copying generational collector with a copying higher generation
- "GenMS" a copying generational collector with a non-copying mark-and-sweep mature space
- "CopyMS" a hybrid non-generational collector with a copying space (into which all allocation goes), and a non-copying space into which survivors go.
- "RefCount" a reference counting collector with synchronous (non-concurrent) cycle collection

For example, to specify a Jikes RVM configuration:

1. with a baseline-compiled boot image,
2. that will compile classes loaded at runtime using the baseline compiler and
3. that uses a non-generational semi-space copying garbage collector,

use the name **`"BaseBaseSemiSpace"`**.

In configurations that include the adaptive system (denoted by **`"Adaptive"`** in their name), methods are initially compiled by one compiler (by default the baseline compiler) and then online profiling is used to automatically select hot methods for recompilation by the optimizing compiler at an appropriate optimization level.

For example, to a build for an adaptive configuration with assertions, where the optimizing compiler is used to compile the boot image and the semi-space garbage collector is used, use the following command:

```
% ant -Dconfig.name=FullAdaptiveSemiSpace
```

Example configurations and their uses

configuration	description	potential uses
BaseBaseSomeGC	baseline compiled bootimage with assertions, baseline compiler at runtime	prototyping; debugging of garbage collector SomeGC without having to worry about complexities introduced by compiler optimizations; checking for problems related to uninterruptible code
BaseAdaptiveSomeGC	baseline compiled bootimage with assertions, baseline compiler, adaptive system and optimizing compiler at runtime	prototyping that includes optimizing compiler and adaptive system; debugging of optimizing compiler problems with compiler advice; sanity checks with comparatively short benchmarks; checking for problems related to uninterruptible code

FullAdaptiveSomeGC	bootimage compiled with optimizing compiler and assertions; everything available at runtime	extensive testing including long-running benchmarks; checking for incorrect usage of unboxed types
ExtremeAssertions*	enables all generally useful assertions, including very expensive ones	debugging and testing in special cases
FastAdaptiveSomeGC	bootimage compiled with optimizing compiler; assertions disabled; everything available at runtime	benchmarking
FullBaseSomeGC	INVALID - Full implies Adaptive	
FastBaseSomeGC	INVALID - Fast implies Adaptive	
ExtremeAssertionsFastAdaptiveSomeGC	INVALID - ExtremeAssertions is incompatible with Fast	

Example mapping of logical configurations to actual configurations

In Jikes RVM 3.1.3, the mapping between logical configurations and actual configurations is:

Logical configuration	Actual configuration
prototype	BaseBaseGenImmix
prototype-opt	BaseAdaptiveGenImmix
development	FullAdaptiveGenImmix
production	FastAdaptiveGenImmix