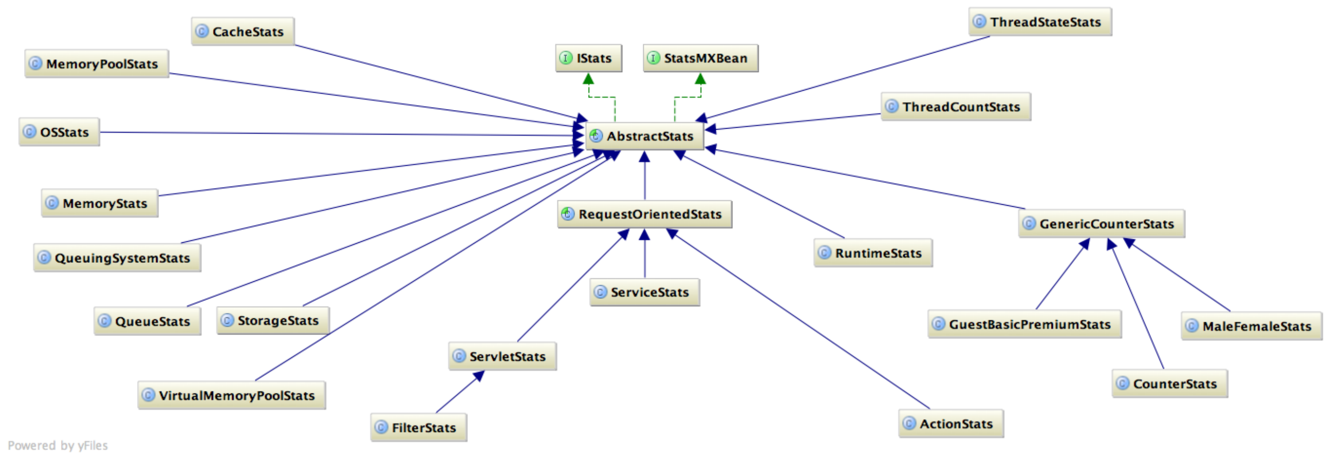
Stats and Their Classes

Generally everything can be monitored with MoSKito. However, some things need to be monitored more often than others. And some things needs to be monitored so often, that they come out of the box. Here a list of Stats objects that come out of the box. Stat objects represents stats (statistics) about something that can be count. A single stat object represents usage of a single monitored object, that can be an **url,** a **method**, a **business case**, a **thread,** a **memory pool,** a **disk** or **file,** a **cpu** or anything else worth monitoring.

Below you will see general stat class and what is monitored.

Stats

The following diagram gives a short overview of stat classes that come out of the box.



Before we begin

How to read the table:

Value	Meaning	Shortcuts	Unit	Notes
name of the value of the appropriate class	What does this attribute actually measure	What shortcuts can be used for generic access to this value	What is the unit (time, amount etc)	Something special to think about?

Please note, even the **Value** is the name of the **StatValue Object used inside the *Stats object,** the accessor methods can differ. For example, to add time to total time in totalRequests you have to call *addExecutionTime.* Also an accessor can do different type of things, for example:

```
        public void addRequest() {
                totalRequests.increase();
                currentRequests.increase();
                maxCurrentRequests.setValueIfGreaterThanCurrentAsLong(currentRequests.getValueAsLong());
        }
        public void addExecutionTime(long time) {
                totalTime.increaseByLong(time);
                lastRequest.setValueAsLong(time);
                minTime.setValueIfLesserThanCurrentAsLong(time);
                maxTime.setValueIfGreaterThanCurrentAsLong(time);
        }
```

Please refer to the documentation and examples on how to use the stat objects correctly (or use built-in annotations) 😉

RequestOrientedStats.

A huge bunch of stats is request oriented (method calls, urls, web-actions etc). The class *net.anotheria.moskito.core.predefined.RequestOrientedStats* provides general functionality for the request oriented stats. The concrete subclasses add only specific handling.

RequestOrientedStats

Value	Meaning	Shortcuts	Unit	Notes
totalRequests	Number of times this request has been executed	TR, REQ	Amount	
totalTime	Total time spent in this request	TT, TIME	Time, NS	
currentRequests	Number of requests *currently* in this request	CR	Amount	
maxCurrentRequests	Maximal recorded number of executions of this request *concurrently*.	MCR	Amount	
errors	Number of recorded errors.	ERR	Amount	
lastRequest	Duration of last request.	LAST	Time, NS	
minTime	*Minimal* duration of this request.	MIN	Time, NS	
maxTime	*Maximal* duration of this request.	MAX	Time, NS	
averageTime	*Average* duration of this request.	AVG	Time, NS	Calculated by totalTime/totalRequests

ServiceStats

Used for service like components (Service, API etc), also for Dynamic Proxies (See *net.anotheria.moskito.core.dynamic.MoskitoInvokationProxy*) and @MonitorClass, @Monitor CDI and AOP annotations.

ServiceStats class has no additional fields.

ActionStats

ActionStats are used by/for action driven frameworks such as struts or ano-maf, but similar to ServiceStats, differs from the RequestStats just by name. The point of having different names(classes) is that it allows the stats to be presented in different decorators and not getting mixed in webui.

ServletStats

ServletStats is the extension of the request oriented stats for http servlets and have three additional values. Each method (get,put,head...) will get its own stats object with all the values. See *net.anotheria.moskito.web.MoskitoHttpServlet* for examples or easy to extend counting servlet.

Value	Meaning	Shortcuts	Unit	Notes
ioExceptions	Number of io exceptions in this method.	IOExc	Amount	
servletExceptions	Number of servlet exceptions in this method.	SEExc	Amount	
runtimeExceptions	Number of runtime exceptions in this method.	RTExc	Amount	

FilterStats

Same as Servlet Stats, but for Filters. See *net.anotheria.moskito.web.MoskitoFilter, net.anotheria.moskito.web.filters.DomainFilter, net.anotheria.moskito.web.filters.RefererFilter or net.anotheria.moskito.web.filters.RequestURIFilter* for usage example or building your own.

Environment and JVM Stats

Environment and JVM Stats are a huge group of stats that are not produced by the Application directly, but by its environment, either the JVM or the operating system or hardware. Most of those stats are gathered from appropriate MBeans.

Memory

Memory is one of the most important parameters of any system in production.

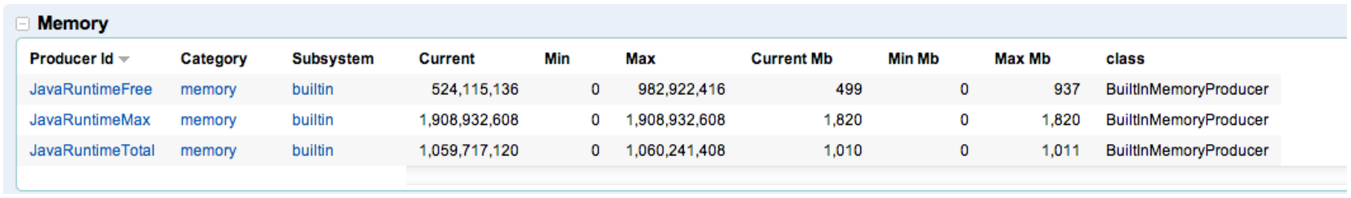
MemoryStats

Based on *Runtime.freeMemory(), maxMemory()* and *totalMemory()* this simple object gives some overview about the memory. MemoryStats are builtin and available ootb. See *net.anotheria.moskito.core.util.BuiltInMemoryProducer.*

A MemoryStats object exists for each stat: free, max and total. The MemoryStats are updated once a minute by a Timer.

Value	Meaning	Shortcuts	Unit	Notes
current	Current amount of memory	CUR	Size, Bytes	
min	Min amount of memory.	MIN	Size, Bytes	
max	Max amount of memory.	MAX	Size, Bytes	

A small example how it looks like in the WebUI:



Memory

Producer Id	Category	Subsystem	Current	Min	Max	Current Mb	Min Mb	Max Mb	class
JavaRuntimeFree	memory	builtin	524,115,136	0	982,922,416	499	0	937	BuiltInMemoryProducer
JavaRuntimeMax	memory	builtin	1,908,932,608	0	1,908,932,608	1,820	0	1,820	BuiltInMemoryProducer
JavaRuntimeTotal	memory	builtin	1,059,717,120	0	1,060,241,408	1,010	0	1,011	BuiltInMemoryProducer

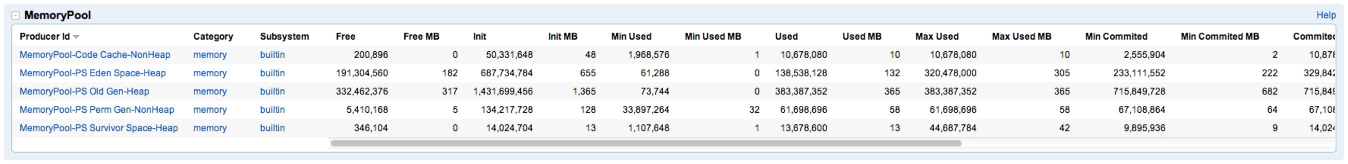
MemoryPoolStats

This stats object is based on the *MemoryPoolMXBean* and provides information about available memory pools. MemoryPoolStats are builtin and available ootb. See *net.anotheria.moskito.core.util.BuiltInMemoryPoolProducer.* MemoryPoolStats are updated once a minute by a Timer.

MemoryPoolStats are much more precise than the MemoryStats. See *java.lang.management.MemoryUsage* for details.

Value	Meaning	Shortcuts	Unit	Notes
init	Initial amount	INIT	Size, Bytes	MemoryUsage.init
used	Used in current interval.	USED	Size, Bytes	MemoryUsage.used
minUsed	Min used in current interval	MIN_USED	Size, Bytes	
maxUsed	Max used in current interval	MAX_USED	Size, Bytes	
commited	Commited in current interval.	COMMITED	Size, Bytes	MemoryUsage.commited
minCommited	Min commited.	MIN_COMMITED	Size, Bytes	
maxCommited	Max commited.	MAX_COMMITED	Size, Bytes	

max	Max available.	MAX	Size, Bytes	MemoryUsage.max

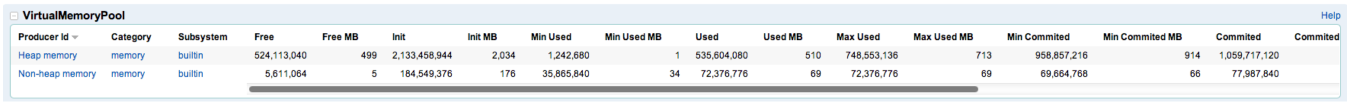


MemoryPool — Help

Producer Id	Category	Subsystem	Free	Free MB	Init	Init MB	Min Used	Min Used MB	Used	Used MB	Max Used	Max Used MB	Min Commited	Min Commited MB	Commite
MemoryPool-Code Cache-NonHeap	memory	builtin	200,896	0	50,331,648	48	1,968,576	1	10,678,080	10	10,678,080	10	2,555,904	2	10,87
MemoryPool-PS Eden Space-Heap	memory	builtin	191,304,560	182	687,734,784	655	61,288	0	138,538,128	132	320,478,000	305	233,111,552	222	329,84
MemoryPool-PS Old Gen-Heap	memory	builtin	332,462,376	317	1,431,699,456	1,365	73,744	0	383,387,352	365	383,387,352	365	715,849,728	682	715,84
MemoryPool-PS Perm Gen-NonHeap	memory	builtin	5,410,168	5	134,217,728	128	33,897,264	32	61,698,696	58	61,698,696	58	67,108,864	64	67,10
MemoryPool-PS Survivor Space-Heap	memory	builtin	346,104	0	14,024,704	13	1,107,648	1	13,678,600	13	44,687,784	42	9,895,936	9	14,02

VirtualMemoryPoolStats

Sometimes its hard to calculate the different heap pools to one heap usage. VirtualMemoryPoolStats do exactly that by combining different pools by their heap-ness into **heap** and **non-heap** pools. VirtualMemoryPoolStats are builtin and available ootb. See *net.anotheria.moskito.core.util. BuiltInMemoryPoolVirtualProducer.* Virtual Memory Pool has exactly the same values (cumulated) as underlying pools and therefore the same values and shortcuts.



VirtualMemoryPool — Help

Producer Id	Category	Subsystem	Free	Free MB	Init	Init MB	Min Used	Min Used MB	Used	Used MB	Max Used	Max Used MB	Min Commited	Min Commited MB	Commited	Commited
Heap memory	memory	builtin	524,113,040	499	2,133,458,944	2,034	1,242,680	1	535,604,080	510	748,553,136	713	958,857,216	914	1,059,717,120	
Non-heap memory	memory	builtin	5,611,064	5	184,549,376	176	35,865,840	34	72,376,776	69	72,376,776	69	69,664,768	66	77,987,840	

Threading

Threads are no less important than memory. The data is based on *java.lang.management.ThreadMXBean*.

ThreadCountStats

This stat object is for simple thread counting. It always has one stat. The data is updated once a minute. See *net.anotheria.moskito.core.util. BuiltInThreadCountProducer.*

Value	Meaning	Shortcut	Units	Note
started	Threads started in the interval.	STARTED	Amount	
current	Currently running threads.	CUR, CURRENT	Amount	
daemon	Currently running daemon threads.	DAEMON	Amount	
minCurrent	Minimum amount of currently running threads.	MIN, MINCURRENT, MIN CUR	Amount	
maxCurrent	Maximum amount of currently running threads.	MAX, MAXCURRENT, MAX CUR	Amount	

ThreadStateStats

ThreadStateStats are ment to hold amount of threads in each state (BLOCKED, WAITING etc). However, the corresponding producer is currently disabled as of 2.0.1. If it would be reenabled, it would deliver following data for each possible STATE:

Value	Meaning	Shortcut	Units	Note
current	Currently running threads in this state.	CUR	Amount	
min	Minimum amount of currently running threads in this state.	MIN	Amount	
max	Maximum amount of currently running threads in this state.	MAX	Amount	

Process

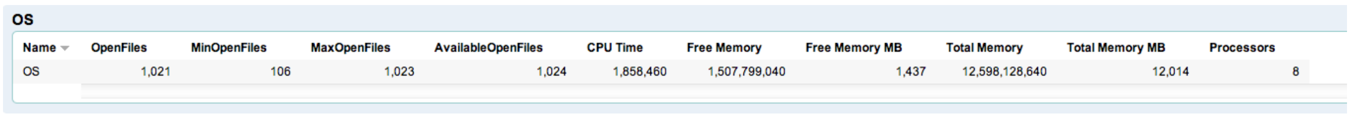
OSStats

OSStats are based on *com.sun.management.UnixOperatingSystemMXBean* and therefore only supported on Unix Platforms.

Value	Meaning	Shortcut	Units	Note
openFiles	Currently open files in the process.	openfiles, open files	Amount	
maxOpenFiles	Max open files by the process.	minopenfiles, min open files	Amount	
minOpenFiles	Min open files by the process.	maxopenfiles, max open files	Amount	

maxSupportedOpenFiles	Max supported open files.	maxsupportedopenfiles	Amount	This value doesn't (shouldn't) change after start.
processCpuTime	CPU Time consumed by this process	cputime, cpu time	Time, Millis	
freePhysicalMemory	Free physical memory.	free memory, free	Size, Bytes	
totalPhysicalMemory	Total physical memory.	total memory, total	Size, Bytes	This value doesn't (shouldn't) change after start.
processors	Number of processors		Amount	This value doesn't (shouldn't) change after start.

Example:



OS

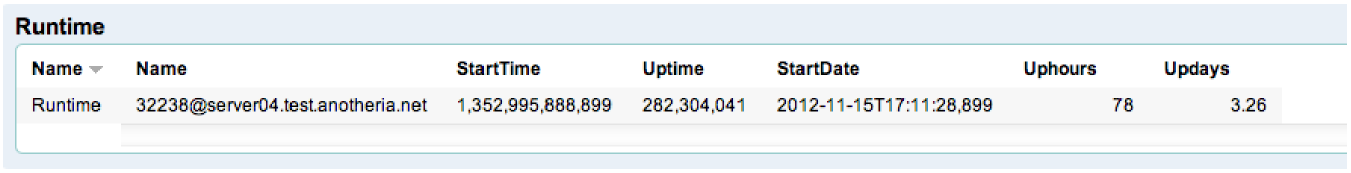
Name	OpenFiles	MinOpenFiles	MaxOpenFiles	AvailableOpenFiles	CPU Time	Free Memory	Free Memory MB	Total Memory	Total Memory MB	Processors
OS	1,021	106	1,023	1,024	1,858,460	1,507,799,040	1,437	12,598,128,640	12,014	8

RuntimeStats

RuntimeStats are useful if you want to know how long your application is running without logging into the machine. They are not *that relevant* for monitoring. It is based on *java.lang.management.RuntimeMXBean.*

Value	Meaning	Shortcut	Units	Note
processName	Currently running threads in this state.	process, name, processname	String	
startTime	Minimum amount of currently running threads in this state.	starttime	Time, Millis since 01.01.1970	
uptime	Maximum amount of currently running threads in this state.	uptime	Time, Millis since start.	

Example:



Runtime

Name	Name	StartTime	Uptime	StartDate	Uphours	Updays
Runtime	32238@server04.test.anotheria.net	1,352,995,888,899	282,304,041	2012-11-15T17:11:28,899	78	3.26

Keeping

Keeping refers to everything that is stored temporary somewhere (probably in memory) and is proceeded somehow. Keeping related stats can be used for capacity planing and monitoring and for throughput monitoring. Following Stats belong into this category:

CacheStats

CacheStats are good for monitoring ... caches! Alas if you want a collection of good and easy to use monitored caches - check out ano-prise-caches.

Not all cache implementations will provide every possible stat value.

Value	Meaning	Shortcut	Units	Note
requests	Get request to the cache.	REQ	Amount	
hits	Hit requests.	HIT	Amount	
writes	Number of writes of new objects to the cache.	WR	Amount	
garbageCollected	Number of items that were garbage collected.	GC	Amount	Only supported by SoftReferenceCaches or similar.
rolloverCount	Number of items that were rolled over.	RO	Amount	Only supported by fixed size caches with rollover. (RoundRobin)
expiredCount	Number of items that were expired.	EX	Amount	Only supported by expiry caches.
filteredCount	Number of items that were in cache but not returned due to filtering	FI	Amount	Only supported by caches that support filtering.
cacheFullCount	Number of times that the cache were full and refused write.	FU	Amount	Only supported by not rolling over caches.
deletes	Number of items removed from the cache.	DEL	Amount	
hitrate	Percentage of hits among gets (hits/requests)	HR		Calculated

QueueStats

A queue in moskitos sense is something between two components, where one component (feeder) add stuff from one end, and the other (processor) works on the other end and removes it. Of course it can be many2many.

Despite their beautifulness, QueueStats are somewhat out of date and are subject to be reworked in near future.

Value	Meaning	Shortcut	Units	Note
requests	Request to put something in the queue.		Amount	
enqueued	Successful request to put something in the queue.		Amount	
dequeued	Successfully processed and therefore removed items.		Amount	
empty	Number of times the processor found the queue empty and had nothing to do.		Amount	
totalSize	Max possible number of elements in the queue.		Amount	
lastSize	Size at the moment of the last operation (which means number of elements in the queue)		Amount	
sumOfSizes	Sum of element count in the queue at updates (puts).		Amount	
maxSize	Max number of elements in the queue.		Amount	
minSize	Min number of elements in the queue.		Amount	

QueuingSystemStats

Used for monitoring of QueuingSystems. As for now, only one such system is known: *net.anotheria.anoprise.processor.QueuedMultiProcessor.*

Despite their beautifulness, QueuingSystemStats are somewhat out of date and are subject to be reworked in near future.

Value	Meaning	Shortcut	Units	Note
serversSize	Amount of servers processing queue.		Amount	
queueSize	Number of elements in the queue.		Amount	
arrived	Number of arrived elements.		Amount	
serviced	Number of serviced elements.		Amount	
errors	Number of errors.		Amount	
waited	Number of elements waiting for processing.		Amount	
throwedAway	Number of elements thrown away due to overflow.		Amount	
waitingTime	Waiting time of the elements.		Time, Millis	
waitingTimeMin	Min waiting time of an element..		Time, Millis	
waitingTimeMax	Max waiting time of an element.		Time, Millis	
servicingTime	Time spent servicing the elements.		Time, Millis	
servicingTimeMin	Min time was spent to serve an element.		Time, Millis	
servicingTimeMax	Max time that was spent to serve an element.		Time, Millis	

StorageStats

Storages are basically wrapper to **maps**. And storage stats monitor accesses to it.

Value	Meaning	Shortcuts	Units	Note
gets	calls to get() method.	G	Amount	
missedGets	gets that returned null	mG	Amount	
puts	calls to put() method	P	Amount	
overwritePuts	calls to put() methods that overwrote an existing value	oP	Amount	
removes	calls to remove() method	RM	Amount	
noopRemoves	calls to remove() method that had no effect (no element there)	noRM	Amount	
size	size of the storage	SZ	Amount	

containsKeyCalls	Calls to containsKey method	CKC	Amount	
containsKeyHits	Calls to containsKey method that returned not null	CKH	Amount	Reason: Calls to containsKey method are expensive, its good to monitor them and their success.
containsValueCalls	Calls to containsValue method	CVC	Amount	
containsValueHits	Calls to containsValue method that returned not null	CVH	Amount	Reason: Calls to containsValue method are **VERY** expensive, its good to monitor them and their success.
missedGets Ratio	missedGets/gets	mG R	Ratio, 0..1	Calculated
hitGets Ratio	(gets-missedGets)/gets	hG R	Ratio, 0..1	Calculated
overwritingPut Ratio	overwritePuts/puts	oP R	Ratio, 0..1	Calculated
newPuts Ratio	(puts-overwritePuts)/puts	nP R	Ratio, 0..1	Calculated
noopRemoves Ratio	noopRemoves/removes	noRM R	Ratio, 0..1	Calculated
PutGet Ratio	puts/gets	PG R	Ratio,	Calculated, if >1 you are putting more elements than you read, this can be unhealthy.
PutRemove Ratio	puts/removes	PRM R	Ratio,	Calculated, if >1 you are putting more elements than you remove, this can be a memory leak.
containsKeyHitRate	containsKeyHits/containsKeyCalls	CK HR	Ratio, 0..1	Calculated
containsKeyValueRate	containsValueHits/containsValueCalls	CV HR	Ratio, 0..1	Calculated

Counters

Counters are basically lightweight producers. Extremely lightweight producers. The have one or multiple dimensions, where each dimension means one stat value. All Counters extend *net.anotheria.moskito.core.counter.GenericCounterStats.*

Counter

The counter is a useful utility for counting stuff. Easy as that. Combined with the *@Count* annotation for both **aop** and **cdi,** Counter allows simple counting of something count-worthy. Therefore it only has one property:

Value	Meaning	Shortcut	Units	Note
counter	Number of countable events.	counter	Amount	

MaleFemale

This is an example of two-dimensional counter and counts separately accesses by male and female users for each case (of course if you ~~let it~~ call it).

Value	Meaning	Shortcut	Units	Note
counter	Number of countable events.	counter	Amount	

GuestMemberPremium

This is an example of a three-dimensional counter.

Value	Meaning	Shortcut	Units	Note
guest	Number of countable events by guest users.	guest	Amount	
member	Number of countable events by members (registered users).	member	Amount	
premium	Number of countable events by premium (paying) users.	premium	Amount	