MoSKito-Essential Configuration Guide

After reading this section, you will know...

... how to configure MoSKito with external configuration file, via ConfigureMe system.

In this section:

Since v2.x, MoSKito may be configured via external configuration file. This configuration is based on ConfigureMe - the state of the art JSON configuration framework.

MoSKito Config is build of different configuration objects, which makes it easier to change and maintain. Each object can be configured separately.

Below is a typical configuration, different sections of it will be discussed separately.

```
{
   "@thresholdsAlertsConfig":{
      "@notificationProviders":[
         {
            "className":"net.anotheria.moskito.core.threshold.alerts.notificationprovider.
LogFileNotificationProvider",
            "parameter":"MoskitoAlert",
            "guardedStatus":"GREEN"
         },
         {
            "className":"net.anotheria.moskito.core.threshold.alerts.notificationprovider.
MailNotificationProvider",
            "parameter":"leon@leon-rosenberg.net",
            "guardedStatus":"RED"
         },
         {
            "className":"net.anotheria.moskito.core.threshold.alerts.notificationprovider.
SysoutNotificationProvider",
            "parameter":"",
            "guardedStatus":"GREEN"
         }
```

```
   ],
   "@alertHistoryConfig":{
      "maxNumberOfItems":500,
      "toleratedNumberOfItems":550
   }
},
"@accumulatorsConfig":{
   "accumulationAmount":500,
   "@accumulators":[
      {
         "name":"Configured SessionCount Cur 5m",
         "producerName":"SessionCount",
         "statName":"Sessions",
         "valueName":"cur",
         "intervalName":"5m"
      }
   ]
},
"@thresholdsConfig":{
   "@thresholds":[
      {
         "name":"Configured-5m-ThreadCount",
         "producerName":"ThreadCount",
         "statName":"ThreadCount",
         "valueName":"Current",
         "intervalName":"5m",
         // time unit can be ignored here
         "@guards":[
            {
               "value":"30",
               "direction":"DOWN",
               "status":"GREEN"
            },
            {
               "value":"30",
               "direction":"UP",
               "status":"YELLOW"
            },
            {
               "value":"45",
               "direction":"UP",
               "status":"ORANGE"
            },
            {
               "value":"60",
               "direction":"UP",
               "status":"RED"
            },
            {
               "value":"100",
               "direction":"UP",
               "status":"PURPLE"
            }
         ]
      }
   ]
},
"@dashboardsConfig":{
   "@dashboards":[
      {
         "name":"Example Dashboard",
         "refresh":60,
         "@charts":[
            {
               "caption":"Threads",
               "accumulators":[
                  "ThreadCount",
                  "ThreadStateBlocked-1m",
                  "ThreadStateRunnable-1m",
                  "ThreadStateTimedWaiting-1m",
                  "ThreadStateWaiting-1m"
```

```
                ]
             },
             {
                "accumulators":[
                   "URL REQ 1m"
                ]
             }
          ],
          "@thresholds":[
             "OrderPerMinuteThreshold",
             "ShopServiceAVG"
          ],
          "@gauges":[
             "Memory",
             "Running",
             "Sessions",
             "SysLoad"
          ],
          "@producers":[
             "ingredients",
             "orders"
          ],
          "@widgets":[
             "gauges",
             "charts",
             "producers",
             "thresholds"
          ],
       }
    ]
  }
}
```

Configuration file and location

MoSKito is configured based on @ConfigureMe Annotations:

```
@ConfigureMe(name="moskito")
public class MoskitoConfiguration {
        @Configure
        private ThresholdsAlertsConfig thresholdsAlertsConfig = new ThresholdsAlertsConfig();
        @Configure
        private ThresholdsConfig thresholdsConfig = new ThresholdsConfig();
        @Configure
        private AccumulatorsConfig accumulatorsConfig = new AccumulatorsConfig();
...
```

ConfigureMe expects the Configuration file to be named **moskito.json** and looks for it in the classpath. However, it is possible to give this file a different name (or use xml or properties instead of json).

```
MoskitoConfiguration configuration = new MoskitoConfiguration();
ConfigurationManager.INSTANCE.configureAs(configuration, "anothername");
MoskitoConfigurationHolder.INSTANCE.setConfiguration(configuration);
```

Of course, you can setup the configuration object entirely by yourself (write the needed code) or get current configuration object from *MoskitoConfigurationHolder* and alter it.
However, do it at the start of the system, since many of the configuration options can't be changed on the fly (yet). Still, you can change others.

Sections

ThresholdsAlerts

The **ThresholdsAlerts** config contains two sections: **AlertHistory** and **NotificationProviders**.

```
        "@thresholdsAlertsConfig": {
                "@notificationProviders": [ NOTIFICATIONPROVIDER ],
                "@alertHistoryConfig": { }
    }
```

AlertHistory

The **AlertHistory** config defines how many items are stored in the in-memory alert history, and can be displayed in MoSKito-WebUI:

```
                "@alertHistoryConfig": {
                        "maxNumberOfItems": 500,
                        "toleratedNumberOfItems": 550
                }
```

Attribute	Value	Default
`maxNumberOfItems`	Number of items that can be stored in the alert history.	200
`toleratedNumberOfItems`	Tolerated overload. The AlertHistory will be shortened only after the size growth above *toleratedNumberOfItems.* This helps to reduce the amount of list operations.	220

NotificationProviders

The built-in notification system allows to configure multiple notification providers. Notification providers will be triggered as soon as a *t hreshold* changes its status and the change breaks the limits of this notification provider.

Each notification provider is configured with the following attributes:

Attribute	Value
className	Name of the class that implements *net.anotheria.moskito.core.threshold.alerts.NotificationProvider*
parameter	Customization of the provider. This attribute is provider-specific.
guardedStatus	The triggering status.

Parameter interpretation:

ClassName	Interpretation
net.anotheria.moskito.core.threshold.alerts.notificationprovider.LogFileNotificationProvider	Name of the Logger.
net.anotheria.moskito.core.threshold.alerts.notificationprovider.MailNotificationProvider	Comma-separated list of recipient's email addresses.
net.anotheria.moskito.core.threshold.alerts.notificationprovider.SysoutNotificationProvider	Ignored.

Thresholds

What is a Threshold?

Thresholds continuously watch a single producer and give a signal when its performance changes.

For more info, read the Thresholds section of MoSKito Concepts.

The thresholds config contains a list of threshold objects. This is another way to define thresholds.

Thresholds may also be added via MoSKito-Inspect.

```
        "@thresholdsConfig": {
                "@thresholds": [ THRESHOLD ]
        },
```

or, in Java words:

```
public class ThresholdsConfig {

        /**
         * Configured thresholds.
         */
        @Configure
        private ThresholdConfig[] thresholds;
```

Threshold

Each Threshold contains the following info:

```
{
  "name": "Configured-5m-ThreadCount",
  "producerName": "ThreadCount",
  "statName": "ThreadCount",
  "valueName": "Current",
  "intervalName": "5m",
  //timeUnit can be ignored here
  "@guards": [ GUARD ]
}
```

Attribute	Value	
name	The name of the value for AlertHistory, Logs and WebUI	
producerName	Name (id) of producer. Exact match required!	
statName	Name of the StatValue. Exact match required!	
valueName	Name of the Value. Exact match required!	
intervalName	Name of the interval the Threshold is tied to.	
timeUnit	TimeUnit if applicable (for example, MILLISECONDS or SECONDS)	See *net.anotheria.moskito.core.stats.TimeUnit*
guards	List of GUARD objects.	

Threshold Guards

A guard is a trigger (set of conditions) that changes the status of a Threshold.

For example:

```
        {"value": "30", "direction": "DOWN", "status": "GREEN"},
        {"value": "30", "direction": "UP", "status": "YELLOW"},
        {"value": "45", "direction": "UP", "status": "ORANGE"},
```

Attribute	Value
value	The value of the associated (producer | stat | statvalue) tuple, which changes the Threshold's status.
direction	Direction in which the value is passed: UP means the current value is higher than the guard value, DOWN - lower than the guard value.

status	The status that the Threshold is set to after the guard is triggered.

Accumulators

> **Accumulators store the performance history of a producer and display accumulated data in charts.**
>
> For more info, read about Accumulators in MoSKito Concepts .

The **accumulators** section configures accumulators, setting the default **accumulationAmount**.

Accumulation amount

```
        "@accumulatorsConfig" : {
                "accumulationAmount": 500
        },
```

The **accumulationAmount** controls the amount of values an accumulator can store. The real amount can be 10% higher, because 10% overload is allowed to reduce number of list operations.

Accumulators

```
        "@accumulatorsConfig" : {
        ...
                "@accumulators": [
                        {
                                "name": "Configured SessionCount Cur 5m",
                                "producerName": "SessionCount",
                                "statName": "Sessions",
                                "valueName": "cur",
                                "intervalName": "5m"
                        }
            ...
                ]
        },
```

Basically, accumulators' section contains the same values for each accumulator as for each threshold:

Attribute	Value	
name	The name of the value for WebUI	
producerName	Name (id) of the producer. Exact match required!	
statName	Name of the StatValue. Exact match required!	
valueName	Name of the Value. Exact match required!	
intervalName	Name of the interval the accumulator is tied to.	
timeUnit	TimeUnit if applicable (for example, MILLISECONDS or SECONDS)	See *net.anotheria.moskito.core.stats.TimeUnit*

AutoAccumulators

AutoAccumulators allow to automatically create accumulator whenever a new producer has been registered which name matches specified pattern:

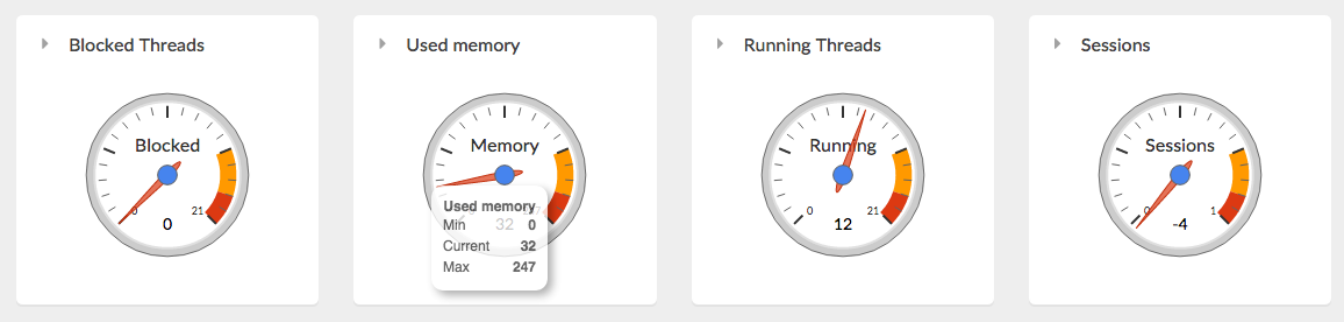
```
"@accumulatorsConfig" : {
  "@autoAccumulators": [
     {
       "namePattern": "$PRODUCERNAME.REQ.1m",
       "producerNamePattern": "(.*)Service",
       "statName": "cumulated",
       "valueName": "req",
       "intervalName": "1m",
       "timeUnit": "MILLISECONDS",
       "accumulationAmount": 0
     },
  ...
}
```

An autoaccumulator section contains the same values as accumulators:

Attribute	Value	
namePattern	The resulting name of the accumulator	It can contain a variable $PRODUCERNAME which will be replaced with actual producerName
producerNamePattern	Pattern for the producer name	Pattern to match, see https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html for example use (.*)Service for all ProducerNames declared as service.
statName	Name of the StatValue. Exact match required!	
valueName	Name of the Value. Exact match required!	
intervalName	Name of the interval the accumulator is tied to.	
timeUnit	TimeUnit if applicable (for example, MILLISECONDS or SECONDS)	See *net.anotheria.moskito.core.stats.TimeUnit*

Gauges

Gauges are a visualization tool for representation of current state of a producer in relation to it's expected min and max states.



Gauges can be used in Dashboards.

Gauges

Gauges are configured in their own section in the configuration file.

```
"@gaugesConfig": {
    "@gauges": [GAUGE],
        "@defaultZones":[ZONE]
}
```

Each gauge is configured in following way

```
{
  "name": "Name of the gauge is displayed in the gauge itself and should be short",
  "caption": "Caption of the gauge block, has more chars to fit",
  "@minValue" : GAUGEVALUE,
  "@currentValue" : GAUGEVALUE,
  "@maxValue" : GAUGEVALUE,
  "@zones": [ZONE]
}
```

A *GAUGEVALUE* can be a constant or a reference to a producer. In the following example a static gauge is configured.

```
"@minValue": {
    "constant": 0
},
"@currentValue": {
  "constant": 70
},
"@maxValue": {
  "constant":100
},
```

One might argue, that a static gauge doesn't make much sense, but it demonstrates the principle and you can use it to present a value which is produced outside of the system.

A GAUGEVALUE can be tight to a producer/stat/value tuple as in following example:

```
{
  "name": "Running",
  "caption": "Running Threads",
  "@minValue": {
    "constant": 0
  },
  "@currentValue": {
    "producerName": "ThreadStates",
    "statName": "RUNNABLE",
    "valueName": "Current",
    "intervalName": "1m"
  },
  "@maxValue": {
    "producerName": "ThreadCount",
    "statName": "ThreadCount",
    "valueName": "current",
    "intervalName": "default"
  }
}
```

Remember you can use either *constant* keyword or *producerName, statName* and *valueName.* If a gauge value config contains *constant* everything else will be ignored for this value.

Besides the values the zones of each gauge can be configured. If you don't provide gauge specific configuration, *defaultZones* are applied. If you provide no *defaultZones* either, the pre-configured default zones are used, which are hardwired in GaugeAPIImpl.

DefaultZones

You can configure default zones which would be applied to all your gauges, if the gauges don't have explicit zone configuration.

```
"@gaugesConfig": {
    "@gauges": [GAUGE],
        "@defaultZones":[ZONE]
}
```

For example:

```
"@defaultZones":[
  {
    "color": "orange",
    "left": 0.85,
    "right": 0.9
  },
  {
    "color": "red",
    "left": 0.9,
    "right": 1
  }
]
```

Full example

Below example configuration of gauges part.

```
"@gaugesConfig": {
    "@gauges": [
      {
        "name": "Constant",
        "@minValue": {
            "constant": 0
        },
        "@currentValue": {
          "constant": 70
        },
        "@maxValue": {
          "constant":100
        },
        "@zones":[
          {
            "color": "green",
            "left": 0,
            "right": 0.25
          },
          {
            "color": "yellow",
            "left": 0.25,
            "right": 0.5
          },
          {
            "color": "orange",
            "left": 0.5,
            "right": 0.75
          },
          {
            "color": "red",
            "left": 0.75,
            "right": 1
          }
        ]

      },
      {
        "name": "Sessions",
        "@minValue": {
```

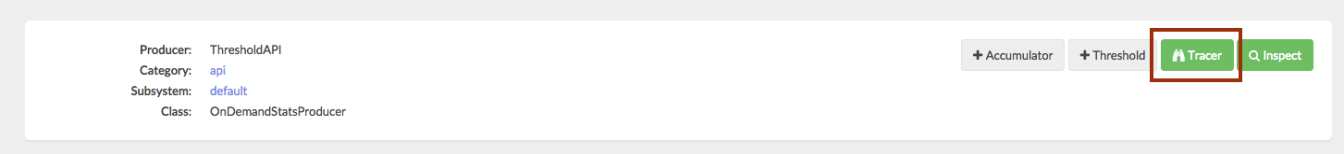
```
      "constant": 0
    },
    "@currentValue": {
      "producerName": "SessionCount",
      "statName": "Sessions",
      "valueName": "cur",
      "intervalName": "default"
    },
    "@maxValue": {
      "producerName": "SessionCount",
      "statName": "Sessions",
      "valueName": "max",
      "intervalName": "default"
    }
  },
  {
    "name": "Memory",
    "caption": "Used memory",
    "@minValue": {
      "constant": 0
    },
    "@currentValue": {
      "producerName": "Heap memory",
      "statName": "Heap memory",
      "valueName": "Used Mb",
      "intervalName": "default"
    },
    "@maxValue": {
      "producerName": "JavaRuntimeMax",
      "statName": "JavaRuntimeMax",
      "valueName": "Current Mb",
      "intervalName": "default"
    }
  },
  {
    "name": "Blocked",
    "caption": "Blocked Threads",
    "@minValue": {
      "constant": 0
    },
    "@currentValue": {
      "producerName": "ThreadStates",
      "statName": "BLOCKED",
      "valueName": "Current",
      "intervalName": "1m"
    },
    "@maxValue": {
      "producerName": "ThreadCount",
      "statName": "ThreadCount",
      "valueName": "current",
      "intervalName": "default"
    }
  },
  {
    "name": "Running",
    "caption": "Running Threads",
    "@minValue": {
      "constant": 0
    },
    "@currentValue": {
      "producerName": "ThreadStates",
      "statName": "RUNNABLE",
      "valueName": "Current",
      "intervalName": "1m"
    },
    "@maxValue": {
      "producerName": "ThreadCount",
      "statName": "ThreadCount",
      "valueName": "current",
      "intervalName": "default"
    }
```

```
      }
    ],
  "@defaultZones":[
    {
      "color": "orange",
      "left": 0.85,
      "right": 0.9
    },
    {
      "color": "red",
      "left": 0.9,
      "right": 1
    }
  ]

},
```

Tracers

Tracers allow you to monitor who is executing a place of code, in code. Tracers are typically switched on/off from inspect on Runtime with the *Tracer* b utton in SingleProducerView in MoSKito Inspect:



However, there are some configuration options for Tracers too.

Tracers are configured via the element *tracingConfig* in MoSKito Configuration.

Here an example:

```
"@tracingConfig": {
  "tracingEnabled": true,
  "loggingEnabled": true,
  "inspectEnabled": true,
  "maxTraces": 50,
  "tracers": [],
  "shrinkingStrategy": "KEEPLONGEST"
}
```

All tracing configuration options are changeable at Runtime. The options mean in particular:

Attribute	Value	
tracingEnabled	true/false. If false tracing won't be active. Tracing can generate some additional load, mainly due to StackTrace creation. So it's wise to switch it off if not needed.	
loggingEnabled	true/false. If true every trace will be logged out into a Logger called *MoSKitoTracer*.	
inspectEnabled	true/false. If true support for inspection in MoSKito Inspect is enabled	
maxTraces	max number of traces (calls with parameters and stacktraces) per Tracer.	To reduce array operations MoSKito will allow Tracers to get 10% more traces than allowed, before cutting them.
tracers	Predefined Tracers. This is basically a list of ProducerIds.	Actually, the idea of tracers is that you want them dynamically, but you can add them in configuration too.

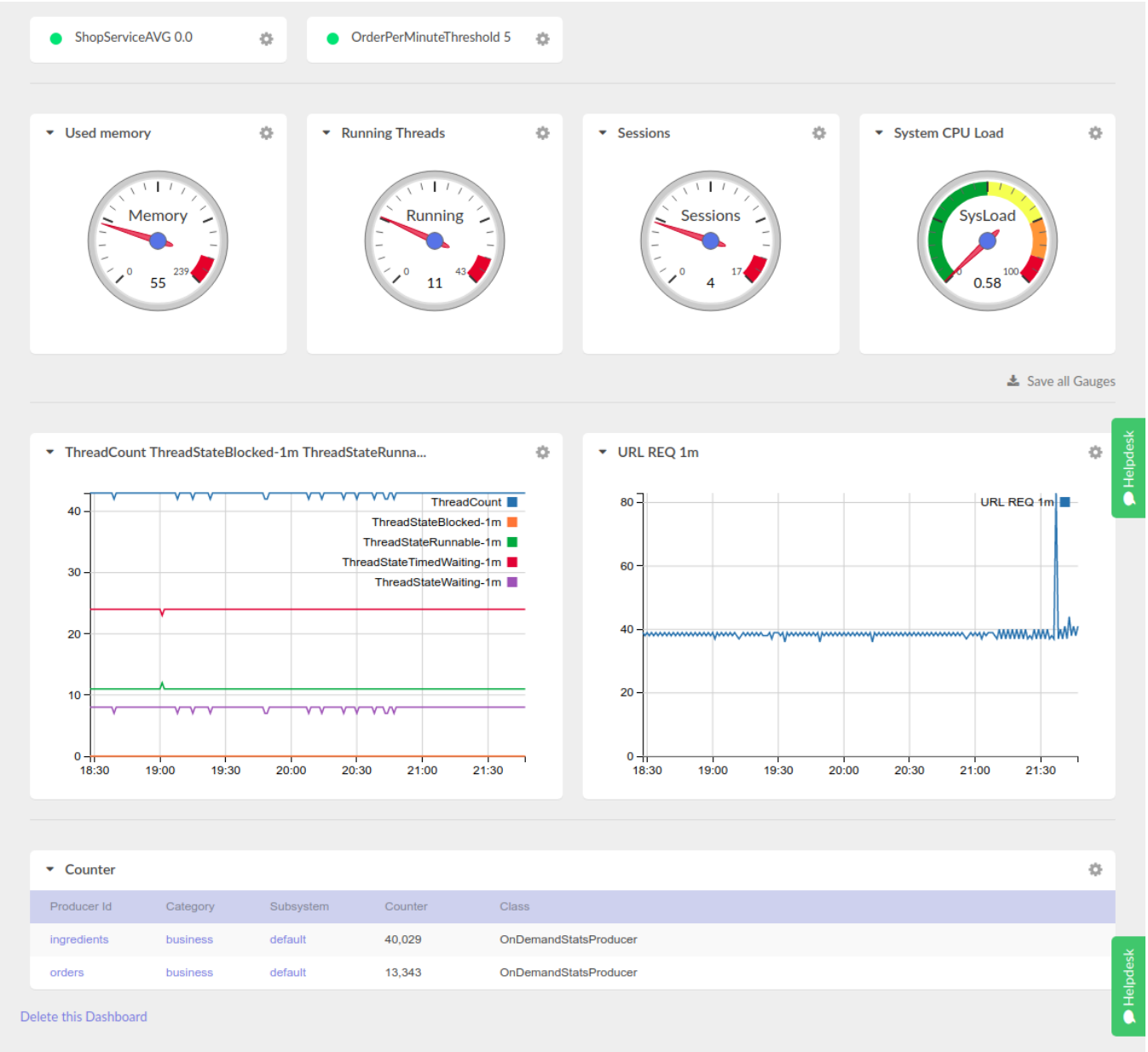
shrinkingStrategy	" FIFO" or " KEEPLONGEST" - defined by *net.anotheria.moskito.core.config.tracing.ShrinkingStrategy*	When amount of tracers exceeds the tolerated amount of traces, MoSKito will start to remove some traces to save memory space. There are two possible strategies here, FIFO -> First in First Out or KEEPLONGEST. Keeplongest sorts the traces by execution duration and keeps those which lasts longer. This is useful in tracking anomalies.

Dashboards

Dashboard allows you to take a look at your system health in general. It's fully customizable and can be changed using appropriate section in configuration file or via MoSKito-Inspect UI. You can create any number of dashboards desired. Dashboards are useful when you don't want to lose sight of key parts of your system.

Each dashboard consists of widgets. Widgets can be of 4 types:

- Thresholds
- Gauges
- Charts
- Producers



Thresholds

In order to display thresholds on dashboard add the following *"@thresholds"* section containing threshold names (exact match is required).

```
"@thresholds":[
  "OrderPerMinuteThreshold",
  "ShopServiceAVG"
]
```

Gauges

Gauges can be added to dashboard by editing appropriate *"@gauges"* section containing gauge names (exact match is required).

```
"@gauges":[
  "Memory",
  "Running",
  "Sessions",
  "SysLoad"
]
```

Charts

Charts section allows you to display accumulated values as line charts.

> For more info, read about Accumulators in MoSKito Concepts .

Each chart has the following properties:

- **Caption** - allows to set chart title, which will be displayed at the top of chart box.
- **Accumulators** - list of accumulator names (exact match is required), in other words - chart lines.

```
"@charts":[
  {
    "caption":"Threads",
    "accumulators":[
      "ThreadCount",
      "ThreadStateBlocked-1m",
      ...
    ]
  },
  ...
]
```

Chart patterns

ChartPattern section allows you to configure accumulators automatically by regular expressions.

> For more info, read about Accumulators in MoSKito Concepts .

Each chart pattern has the following properties:

- **Caption** - allows to set chart title, which will be displayed at the top of chart box.
- **Accumulator patterns** - a list of strings, each of them a regex pattern. All accumulators which names matches the pattern will be added to a chart.
- **Mode** - either "COMBINE" or "MULTIPLE".
  If the mode is MULTIPLE that caption is ignored and every accumulator that matches the pattern is added to the dashboard.
  If the mode is COMBINE all accumulators matching the pattern are added in one chart and the caption is used above the chart.

```
"@chartPatterns":[
  {
    "caption":"Threads",
    "accumulatorPatterns":[
      "ThreadStateBlocked(.*)",
      "ThreadStateWaiting(.*)",
      ...
    ],
                "mode": "MULTIPLE"
  },
  ...
]
```

Producers

Producers can be added to dashboard by editing appropriate *"@producers"* section. Just list producer names (exact match is required). Producers will be grouped by decorators.

```
"@producers":[
  "ingredients",
  "orders"
]
```

Producer name patterns

This sections allows you to configure producers automatically by regular expressions. All producers which names (ids) matches the pattern will be added to the dashboard. Producers will be grouped by decorators.

```
"@producerNamePatterns":[
  "(.*) memory",
  "(.*)Service(.*)"
]
```

Configure widgets

Order and presence of widgets can be configured in *"@widgets"* section for each dashboard. Order in array determines display order on dashboard and array entries determine presence of widgets on dashboard. Possible widget names: *"thresholds"*, *"gauges"*, *"charts"*, *"thresholds"*.

```
"@widgets":[
  "gauges",
  "charts",
  "producers"
]
```

If there is no widget section specified in dashboard config section, default configuration is used, which is

```
"@widgets":[
  "thresholds",
  "gauges",
  "charts",
  "producers"
]
```

Plugins

Plugins section allows to load custom software aka plugins.

```
        "@pluginsConfig": {
                "@plugins": [
                        {
                                "name": "EmbeddedCentralConnector",
                                "configurationName": "none",
                                "className": "net.anotheria.moskito.central.connectors.embedded.
EmbeddedConnector"
                        }
                ]
        }
```

For each plugin, the following values are configured:

Attribute	Value	
name	The name of the plugin for plugin view.	
configurationName	The name of the plugin configuration.	The configuration is of plugin-special type.
className	The name of the plugin class.	The class should implement *net.anotheria.moskito.core.plugins.Moskitoplugin*

Builtin Producers

Builtin Producers section allows to configure which builtin producers should be enabled by default. If you don't set anything, all producers are enabled (default value = true).

Example:

```
        "@builtinProducersConfig": {
                "javaMemoryProducers": false,
                "javaMemoryPoolProducers": false,
                "osProducer": false
        }
```

Supported attributes are:

Attribute	Producers
javaMemoryProducers	Memory based on Runtime.getRuntime().freeMemory
javaMemoryPoolProducers	Memory based on GC Memory Pools / Spaces
javaThreadingProducers	ThreadCountProducer ThreadStatesProducer
osProducer	OS Stats (on *nix only) incldung min/max files etc
runtimeProducer	Runtime - process name and uptime
gcProducer	Creates a gc producer for every garbage collector mbean found in system

Additional configurations are required for MBeans producers and Tomcat Global Request Processor producer.

MbeanProducerConfig example:

```
"@mbeanProducersConfig": {
  "registerAutomatically": true,
  "updateAutomatically": true,
  "delayBeforeFirstUpdate": 15000,
  "@domains": [
    {
      "name": "java.lang",
      "classes": [
        "sun.management.ClassLoadingImpl"
      ]
    }
  ]
}
```

Supported attributes are:

Attribute	Producers
registerAutomatically	Indicates if the producers will be registered automatically. Defaults is false.
updateAutomatically	Indicates if the producer values will be updated automatically. Defaults is true.
delayBeforeFirstUpdate	Time in milliseconds before start the first producer values update. Defaults is 15 seconds.
domains	List of all Domains, null means that producers will be created for all MBean domains. Defaults is null. • name - the name of MBean domain to configure • classes - a list of classes which will be registered as producer, null means all MBeans in that domain will be registered. Defaults is null.

TomcatRequestProcessorProducerConfig example:

```
"@tomcatRequestProcessorProducerConfig": {
  "ajp": true,
  "http": true
}
```

Supported attributes are:

Attribute	Producers
ajp	Indicates that ajp processor should be monitored. Defaults is true
http	Indicates that http processor should be monitored. Defaults is true

Journey

Journey are configured in their own section in the configuration file.

```
"@journeyConfig": {
    "parameterLengthLimit": 200,
        "toStringCollections": false,
        "toStringMaps": false
 }
```

Each journey is configured in following way

Attribute	Value	Default
parameterLengthLimit	Limit for the parameter length. If parameter length is more than configured, it will be cut. This also applies to return values.	100
toStringCollections	If true collections are "toStringed", if false, only size is shown.	true
toStringMaps	If true maps are "toStringed", if false, only size is shown.	true

Errorhandling

Errorhandling config defines how MoSKito stores and helps you to handle errors that occur in your application.

Here is a configuration example:

```
"@errorHandlingConfig": {
  "autoChartErrors": true,
  "autoChartErrorsInterval": "1m",
  "logErrors": true,
  "@catchers": [
    {
      "exceptionClazz": "java.lang.IllegalArgumentException",
      "target": "LOGANDMEMORY",
      "parameter": "IllegalArgExcLog"
    },
    {
      "exceptionClazz": "org.springframework.web.util.NestedServletException",
      "target": "MEMORY",
      "parameter": ""
    }
  ],
  "catchersMemoryErrorLimit": 50
```

Attribute	Meaning
autoChartErrors	If true, for every throwable that MoSKito *sees* an accumulator will be created.
autoChartErrorsInterval	Name of the interval the error accumulators have to use. This is used for the *autoChartedErrors* (accumulators that have been created if *autoChartErrors=true)* as well as *cumulated* errors.
logErrors	If true, all errors will be logged to the log-file named *auto-caught-errors.log.* The logger name can be changed in the logback.xml (or any other slf4j compartible logging framework you are using).
catchersMemoryErrorLimit	If catchers are configured, they will hold max 10% more last errors, that the value of catchersMemoryErrorLimit. Default is 50.
countRethrows	If countRethrows is enabled MoSKito will try to analyze if errors are rethrown within the execution. Default is false. This is due to the fact, that the rethrows are saved in a ThreadLocal variable within the MoSKitoContext and could lead to memory leak if *MoSKitoContext.cleanup()* is not called after the execution.
@catchers	Defined catchers for specific errors.

Catchers

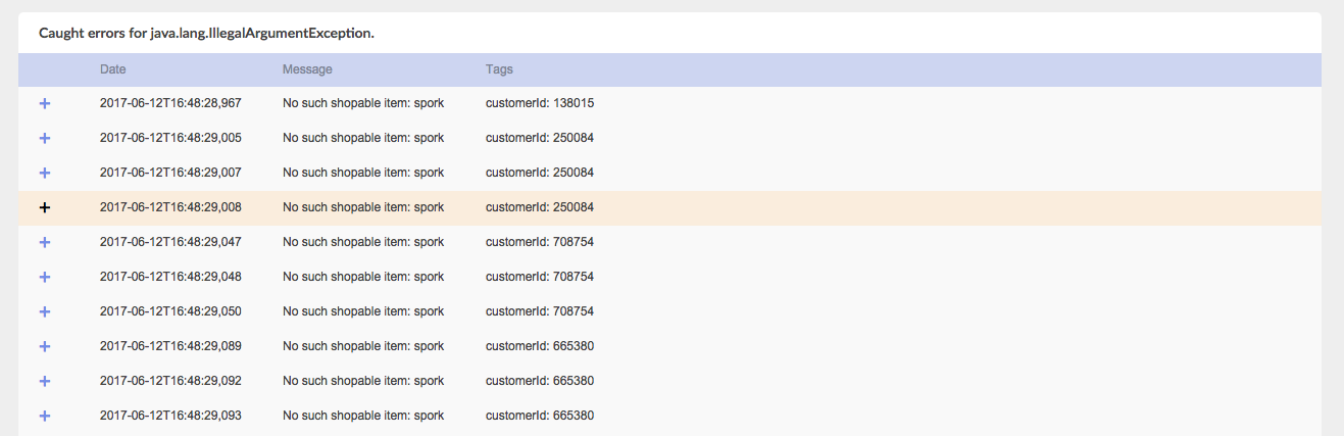
You can configure one or multiple catchers for specific exceptions you want to investigate.

Attribute	Meaning

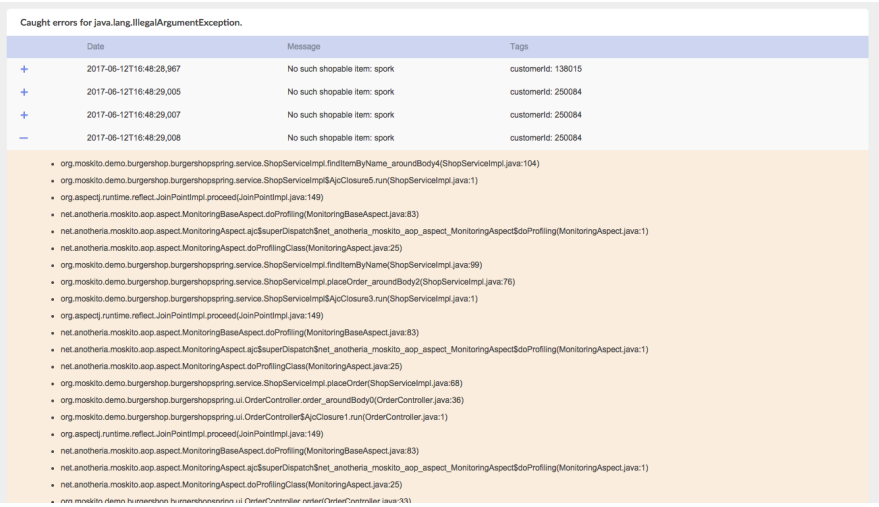
`exceptionClazz`	Exception class, for example *java.lang.IllegalArgumentException*
target	Catch target, is one of: LOG, MEMORY or LOGANDMEMORY. Memory means that the stacktraces will be held in memory for investigation via console. Log means that the exceptions will be logged into a specific logfile.
`parameter`	In case of LOG or LOGANDMEMORY target name of the logger where the exception has to be send. You will need to configure this logger separately in the logback.xml

Catchers will are accessible from the Navigation point: Everything else | Errors and look like this:

Every catcher will container up to *catchersMemoryErrorLimit* errors:

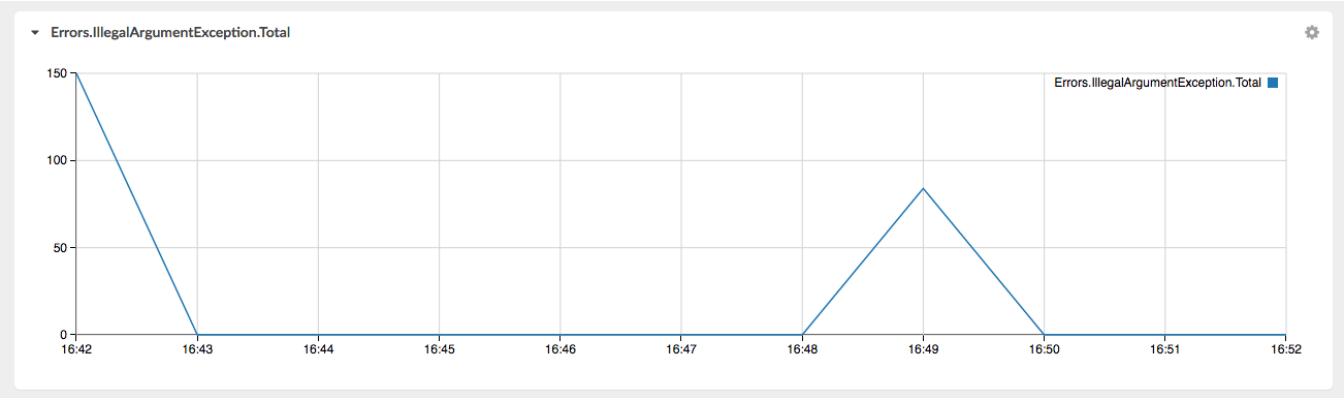


And you can click on every error to see the stack trace:



Autocharted errors

In case autocharting is enabled two charts will be created for each error, a total and initial value. Initial means errors which were the first in the execution thread, total - all errors, including followup errors.



this allows you to analyse error occurrence visually.

Tagging

Tagging allows you to add additional information to journeys and caught errors. An example tagging config looks like this:

```
"taggingConfig": {
  "autotagSessionId": true,
  "autotagIp": true,
  "autotagUserAgent": true,
  "autotagReferer": true,
      "autotagUrl": true,
      "autotagServerName": true,
  "@customTags": [
    {
      "name": "CustomerID",
      "attribute": "session.customerId"
    }
  ],
  "tagHistorySize": 10
}
```

Builtin or Autotags.

MoSKito will automatically tag *session, ip, referrer, user-agent, url and server name* unless you disable them by setting the corresponding property to false in the tagging config. Tagging has been added in MoSKito 2.8.4.

Custom Tags

MoSkito can automatically use any header, parameter, request or session attribute value as tag. To configure a custom tag, you have to specify a name and path as json object under *@customTags* element:

```
   {
      "name": "CustomerID",
      "attribute": "session.customerId"
    }
```

Where name is everything you want and attribute consists of a *prefix* and *attributename.* Following prefixes are supported:

- header
- request
- session
- parameter

See CustomTagSource for details.

Tag History

For debugging and informational purposes MoSKito keeps a number of elements in the history for every tag. This way you can see what were the last seen value and check if they meet your expectation. Default number of values is 10.