README.md

```markdown
# Dynamic Tagged User Metrics

#### Vector

## Meters
There are three (3) types of meters implemented by Micrometer:
1. Counter - counts the number of times an activity has been performed.
2. Gauge - measure a point in time for an activity.
3. Timer - records the time an activity took to execute.

### Tagged Meters
Tagged meters are Micrometer counters, gauges and timers that are created using a single tag name which act
like a category for the meter.

### Tagged Counter

Counters monitor monotonically increasing values. Counters may never be reset to a lesser value.
If you need to track a value that goes up and down, use a Gauge.

##### Create tagged counter

    Create a tagged micrometer counter
        @param name meter name
        @param description meter description
        @param tagName tag name associated with timer
        @param tagValue tag value associated with timer

    createTagCounter(String name, String description, String tagName, String tagValue)


##### Increment tagged counter by value

    Increments a defined tagged counter with supplied incrementsl value
        @param name meter name
        @param increment incremental value

    incrementTagCounter(String name, long increment)

##### Increment tagged counter by 1

    Increments a defined tag counter
        @param name meter name

    incrementTagCounter(String name)

##### Example

    Create Tagged Counter: createTagCounter(name="dogs",description="dog counter", tagName="breed", tagValue="lab")

    Increment Tagged Counter (BY 1): incrementTagCounter(name="dogs")

    Increment Tagged Counter (BY 20): incrementTagCounter(name="dogs", value=20)

### Tagged Gauge

A gauge tracks a value that may go up or down.
The value that is published for gauges is an instantaneous sample of the gauge at publishing time.

##### Create tagged gauge

    Creates a tagged micrometer gauge
        @param name meter name
        @param description meter description
        @param tagName tag name associated with timer
        @param tagValue tag value associated with timer

    createTagGauge(String name, String description, String tagName, String tagValue)

##### Set tagged gauge value

    Set the value of a defined tagged gauge
        @param name meter name
        @param value the value to set the gauge

    setTagGauge(String name, double value)

#### Example

    Create Tagged Gauge: createTagGauge(name="dogs",description="dog gauge",tagName="DogsInKennel", tagValue="lab")

    Set Tagged Gauge: setTagGauge(name="dogs", value=60d)

### Tagged Timer

Timer intended to track of a large number of short running events. Example would be something like an HTTP request.
Though "short running" is a bit subjective the assumption is that it should be under a minute.

##### Create tagged timer

    Creates a tagged micrometer timer
        @param name meter name
        @param description meter description
        @param tagName tag name associated with timer
        @param tagValue tag value associated with timer

    createTagTimer(String name, String description, String tagName, String tagValue)

##### Create tagged histogram timer

    Creates a histogram timer
        @param name meter name
        @param description meter description
        @param tagName tag name associated with timer
        @param tagValue tag value associated with timer

    createTagHistogramTimer(String name, String description, String tagName, String tagValue)

##### Create tagged histogram timer with expected values

    Creates a histogram timer with expected values
        @param name meter name
        @param description meter description
        @param type timer type
        @param min minimum expected value
        @param max maximum expected value
        @param tagName tag name associated with timer
        @param tagValue tag value associated with timer

    createTagHistogramExpectedValueTimer(String name, String description, TimerTypes.TimerType type,
        Duration min, Duration max, String tagName, String tagValue)

##### Create tagged histogram timer with expected values and SLO/SLA

    Creates a histogram timer with expected value and service level objective or service level agreement
        @param name meter name
        @param description meter description
        @param type timer type
        @param slosla duration for SLA/SLO
        @param min minimum expected value
        @param max maximum expected value
        @param tagName tag name associated with timer
        @param tagValue tag value associated with timer

        @throws InvalidOptionType invalid timer type specified

    createTagHistogramExpectedValueSloSlaTimer(String name, String description, TimerTypes.TimerType type,
        Duration slosla, Duration min, Duration max, String tagName, String tagValue) throws InvalidOptionType
```

##### Record tagged time with duration

```
Records the time for a tagged timer
    @param name meter name
    @param duration duration of time to record

recordTagTime(String name, Duration duration)
```

##### Record tagged time with time unit and value

```
Records the time for a tagged timer
    @param name meter name
    @param timeUnit time unit of the time value
    @param timeValue time value

    @throws InvalidTimeUnitException invalid time unit specified

recordTagTime(String name, TimeUnit timeUnit, long timeValue) throws InvalidTimeUnitException
```

#### Example

```
Create Tagged Timer: createTagTimer(name="dogs",description="dog timer",tagName="checkin", tagValue="lab")

Record Tagged Timer: recordTagTime(name="dogs", duration=Duration.ofSeconds(27))

Record Tagged Timer: recordTagTime(name="dogs", timeUnit=TimeUnit.Seconds, timeValue=27)
```

### Multi-tagged Meters

Multi-tagged meters are Micrometer counters, gauges and timers that are created using a set of tag names which act like a set of categories for the meter.

### Multi-tagged Counter

Counters monitor monotonically increasing values. Counters may never be reset to a lesser value. If you need to track a value that goes up and down, use a Gauge.

##### Create multi-tagged counter

```
Creates a multi-tag micrometer counter
    @param name meter name
    @param description meter description
    @param tagNames string array of tag names associated with timer
    @param tagValues string array of tag values associated with timer

CreateMultiTagCounter(String name, String description, String[] tagNames, String[] tagValues)
```

##### Increment multi-tagged counter by 1

```
Increments multi-tag counter
    @param name meter name

incrementMultiTagCounter(String name)
```

##### Increment multi-tagged counter by an incremental value

```
Increments a multi-tag counter with an increment value
```

```
    @param name meter name
    @param increment increment number

incrementMultiTagCounter(String name, long increment)
```

##### Example

```
Create MultiTagged Counter: CreateMultiTagCounter(name="groups", description="team groups names and colors", tagNames="team color", tagValues="vector blue")

Increment MultiTagged Counter (BY 1): incrementMultiTagCounter(name="groups")

Increment MultiTagged Counter (BY 20): incrementMultiTagCounter(name="groups", value=20)
```

### Multi-tagged Gauge

A gauge tracks a value that may go up or down.
The value that is published for gauges is an instantaneous sample of the gauge at publishing time.

##### Create multi-tagged gauge

```
Creates a multi-tagged tagged micrometer gauge
    @param name meter name
    @param description meter description
    @param tagNames string array of tag names associated with timer
    @param tagValues string array of tag values associated with timer

    @throws IllegalArgumentException tagName and tagValue array lengths are unequal

createMultiTagGauge(String name, String description, String[] tagNames, String[] tagValues) throws IllegalArgumentException
```

##### Set multi-tagged gauge value

```
Set the value of a defined multi-tagged gauge
    @param name meter name
    @param value the value to set the gauge

setMultiTagGauge(String name, double value)
```

#### Example
```
Create MultiTagged Gauge: createMultiTagGauge(name="weekly-high-tides", description="weekly high tide gauge", tagNames="city, day", tagValues="austin monday")

Set MultiTagged Gauge: setMultiTagGauge(name="weekly-high-tides", value=1.53d)
```

### Multi-tagged Timer

Timer intended to track of a large number of short running events. Example would be something like an HTTP request.
Though "short running" is a bit subjective the assumption is that it should be under a minute.

##### Create multi-tagged timer

```
Create multi-tagged timer
    @param name meter name
    @param description meter description
    @param tagNames string array of tag names associated with timer
    @param tagValues string array of tag values associated with timer

createMultiTagTimer(String name, String description, String[] tagNames, String[] tagValues)
```

##### Create multi-tagged histogram timer

```
Creates a histogram timer
    @param name meter name
    @param description meter description
    @param tagNames string array of tag names associated with timer
    @param tagValues string array of tag values associated with timer
    createMultiTagHistogramTimer(String name, String description, String[] tagNames, String[] tagValues)

##### Create multi-tagged histogram timer with expected values

    Creates a histogram timer with expected values
        @param name meter name
        @param description meter description
        @param type timer type
        @param min minimum expected value
        @param max maximum expected value
        @param tagNames string array of tag names associated with timer
        @param tagValues string array of tag values associated with timer

    createMultiTagHistogramExpectedValueTimer(String name, String description, TimerTypes.TimerType type,
        Duration min, Duration max, String[] tagNames, String[] tagValues)

##### Create multi-tagged histogram timer with expected values and SLO/SLA

    Creates a histogram timer with expected value and service level objective or service level agreement
        @param name meter name
        @param description meter description
        @param type timer type
        @param slosla duration for SLA/SLO
        @param min minimum expected value
        @param max maximum expected value
        @param tagNames string array of tag names associated with timer
        @param tagValues string array of tag values associated with timer

        @throws InvalidOptionType invalid timer type specified

    createTagHistogramExpectedValueSloSlaTimer(String name, String description, TimerTypes.TimerType type,
        Duration slosla, Duration min, Duration max, String[] tagNames, String[] tagValues) throws InvalidOptionType

##### Record multi-tagged timer with duration

    Records the time for a multi-tagged timer with duration
        @param name meter name
        @param duration duration of time to record

    recordMultiTagTime(String name, Duration duration)

##### Record multi-tagged timer with time unit and value

    Records the time for a multi-tagged timer with timeunit and value
        @param name meter name
        @param timeUnit time unit of the time value
        @param timeValue time value

        @throws InvalidTimeUnitException - invalid time unit specified

    recordMultiTagTime(String name, TimeUnit timeUnit, long timeValue) throws InvalidTimeUnitException

#### Example

    Create MultiTagged Timer: createMultiTagTimer(name="action-timer",description="action timer",tagNames="who, action", tagValues="vector stream")

    Record MultiTagged Timer: recordMultiTagTime(name="action-timer", duration=Duration.ofMinutes(27))

    Record MultiTagged Timer: recordMultiTagTime(name="action-timer", timeUnit=TimeUnit.Seconds, timeValue=27l)

### Dependencies

#### Spring

Spring Boot Actuator uses HTTP endpoints to expose operational information about any running application.
```

```
    org.springframework.boot.spring-boot-starter-actuator:{version from spring boot}
```

#### Prometheus ####

```
    io.prometheus.simpleclient:0.16.0

    io.prometheus.simpleclient_common:0.16.0

    io.prometheus.simpleclient_httpserver:0.16.0
```

#### Micrometer ####

```
    io.micrometer.micrometer-core:1.10.6

    io.micrometer.micrometer-registry-prometheus:1.10.6
```

## MetricHelper

The MetricHelper class provides the implementation for both tagged and multi-tagged meters. The implementation allows application teams the capability to enable metrics in their Spring Boot applications and have these metrics exposed for scrapping by Prometheus.

Refer to the following link for MetricHelper javadoc:

[MetricHelper class](./MetricHelper.pdf)

## Spring Micrometer Annotations

Spring provides two (2) out-of-box annotations (@Timed and @Counted) for Micrometer metrics. These annotations can only be used when instrumenting a method to determine the number of times the method was called and the time it took to perform the method.

The @Counted annotation supports four (4) fields/properties:

```
    1. value: name of the meter
    2. recordFailuresOnly: count only failures
    3. description: description of the meter
    4. extraTags: extra tags to further qualify or categorize the meter
```

The @Timed annotation supports six (6) fields/properties:

```
    1. value: name of the meter
    2. description: description of the meter
    3. longTask: the method/task is long-running
    4. percentiles: defines the percentile buckets to be used
    5. histograms: include histograms
    6. extraTags: extra tags to further qualify or categorize the meter
```

The following is an example on how to use these annotations:

```
        @RestController
        public class MyController {
                @Timed(value = "example-timer", description = "captures the time to perform the rest request", longTask = false, histogram = true, extraTags = {})
                @Counted(value = "example-counter", description = "captures the number of times the rest service was called", recordFailuresOnly = false, extraTags = {})
                @GetMapping("/example")
                public String get(HttpServletRequest servletRequest) {
                        // do work
                }
        }
```

## Spring Boot

### Annotation

To properly wire the MetricHelper requires adding a component scan on the Spring Boot main class.

```
    @SpringBootApplication
    @ComponentScan(value = "mil.army.swf.micrometer.metrics.*")
```

```
public class ExampleMetricsApplication {
    public static void main(String[] args) {
        SpringApplication.run(ExampleMetricsApplication.class, args);
    }
}
```

## Configuration

In order for prometheus to scrape the server for metrics, requires configuration in the application yaml or properties file.

### YAML/Property

#### YAML

```
management:
  endpoint:
    prometheus:
      enabled: true
  endpoints:
    web:
      exposure:
        include: '*'
      base-path: /actuator
```

#### Property

```
management.endpoint.prometheus.enabled=true
management.endpoints.web.exposure.include='*'
management.endpoints.web.base-path='/actuator'
```