AT&T Service Specification

Addendum: *VES Event Registration*

|  |  |
| --- | --- |
| Document Number | xxxx |
| Revision | 1.4 |
| Revision Date | April 14, 2017 |
| Author | Rich Erickson |

This document and the information contained herein (collectively, the "**Information**") is provided to you (both the individual receiving this document and any legal entity on behalf of which such individual is acting) ("**You**" and "**Your**") by AT&T, on behalf of itself and its affiliates ("**AT&T**") for informational purposes only. AT&T is providing the Information to You because AT&T believes the Information may be useful to You. The Information is provided to You solely on the basis that You will be responsible for making Your own assessments of the Information and are advised to verify all representations, statements and information before using or relying upon any of the Information. Although AT&T has exercised reasonable care in providing the Information to You, AT&T does not warrant the accuracy of the Information and is not responsible for any damages arising from Your use of or reliance upon the Information. You further understand and agree that AT&T in no way represents, and You in no way rely on a belief, that AT&T is providing the Information in accordance with any standard or service (routine, customary or otherwise) related to the consulting, services, hardware or software industries.

AT&T DOES NOT WARRANT THAT THE INFORMATION IS ERROR-FREE. AT&T IS PROVIDING THE INFORMATION TO YOU "AS IS" AND "WITH ALL FAULTS." AT&T DOES NOT WARRANT, BY VIRTUE OF THIS DOCUMENT, OR BY ANY COURSE OF PERFORMANCE, COURSE OF DEALING, USAGE OF TRADE OR ANY COLLATERAL DOCUMENT HEREUNDER OR OTHERWISE, AND HEREBY EXPRESSLY DISCLAIMS, ANY REPRESENTATION OR WARRANTY OF ANY KIND WITH RESPECT TO THE INFORMATION, INCLUDING, WITHOUT LIMITATION, ANY REPRESENTATION OR WARRANTY OF DESIGN, PERFORMANCE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, OR ANY REPRESENTATION OR WARRANTY THAT THE INFORMATION IS APPLICABLE TO OR INTEROPERABLE WITH ANY SYSTEM, DATA, HARDWARE OR SOFTWARE OF ANY KIND. AT&T DISCLAIMS AND IN NO EVENT SHALL BE LIABLE FOR ANY LOSSES OR DAMAGES OF ANY KIND, WHETHER DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, PUNITIVE, SPECIAL OR EXEMPLARY, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF BUSINESS INFORMATION, LOSS OF GOODWILL, COVER, TORTIOUS CONDUCT OR OTHER PECUNIARY LOSS, ARISING OUT OF OR IN ANY WAY RELATED TO THE PROVISION, NON-PROVISION, USE OR NON-USE OF THE INFORMATION, EVEN IF AT&T HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH LOSSES OR DAMAGES.

© 2015 AT&T Intellectual Property

All rights reserved.

AT&T and AT&T logos are trademarks of AT&T Intellectual Property.

All marks, trademarks, and product names used in this document are the property of their respective owners.

Version 1.0 Change Log (for historical change logs, see the Appendix)

| Date | Revision | Description |
| --- | --- | --- |
| 4/14/2017 | 1.4 | * Wordsmithed throughout * Action keyword: clarified use of ‘up’, ‘down’ and ‘at’ triggers; clarified the specification and use of microservices directives at design time and runtime, clarified the use of tca’s * HeartbeatAction keyword: Added the heartbeatAction keyword * Value keyword: clarified the communicaton of strings containing spaces. * Rules: corrected the use of quotes in examples * Examples: added the heartbeatAction keyword on the heartbeat event example; also corrected use of quotes throughout. |

[1 Introduction 1](#_Toc479937565)

[1.1 Audience 1](#_Toc479937566)

[1.2 Goal 1](#_Toc479937567)

[1.3 Relation to the Common Event Format 1](#_Toc479937568)

[1.4 Relation to Service Design and Creation 2](#_Toc479937569)

[2 YAML Files 3](#_Toc479937570)

[2.1 YAML Specification Conformance 3](#_Toc479937571)

[2.2 Filename 3](#_Toc479937572)

[2.3 File Structure 3](#_Toc479937573)

[3 YAML Syntax and Semantics 4](#_Toc479937574)

[3.1 Qualifiers 4](#_Toc479937575)

[3.1.1 Action 4](#_Toc479937576)

[3.1.2 Array 5](#_Toc479937577)

[3.1.3 Default 6](#_Toc479937578)

[3.1.4 HeartbeatAction 6](#_Toc479937579)

[3.1.5 Presence 6](#_Toc479937580)

[3.1.6 Range 7](#_Toc479937581)

[3.1.7 Structure 7](#_Toc479937582)

[3.1.8 Value 7](#_Toc479937583)

[3.2 Rules 8](#_Toc479937584)

[3.2.1 Rules Document 8](#_Toc479937585)

[3.2.2 Rules Syntax and Semantics 8](#_Toc479937586)

[3.2.3 Simple Triggers 9](#_Toc479937587)

[3.2.4 Time Based Qualifiers 10](#_Toc479937588)

[4 YAML Examples 11](#_Toc479937589)

[5 Appendix: Historical Change Log 15](#_Toc479937590)

# Introduction

This document specifies a YAML format for the registration of VES Events. The YAML format enables both human designers and applications to parse and understand the fields that will be sent by event sources in conjunction with specific types of events, which are identified by their eventNames.

The semantics of the YAML format are easily extensible to accommodate processing needs that may arise in the future. Among the types of information specified in the YAML are field optionality, restrictions on field values, and event handling recommendations and requirements.

This document should be read in conjunction with the VES Event Listener service specification, which defines the Common Event Format and introduces the concept of specific types of events, identified by eventNames.

## Audience

This document is intended to support the following groups:

* VNF Vendors
* Service Provider (e.g., AT&T) Teams responsible for deploying VNFs within their infrastructure

VNF vendors will provide a YAML file to the Service Provider that describes the events that their VNFs generate. Using the semantics and syntax supported by YAML, vendors will indicate specific conditions that may arise, and recommend actions that should be taken at specific thresholds, or if specific conditions repeat within a specified time interval.

Based on the vendor’s recommendations, the Service Provider may create another YAML, which finalizes their engineering rules for the processing of the vendor’s events. The Service Provider may alter the threshold levels recommended by the vendor, and may modify and more clearly specify actions that should be taken when specified conditions arise. The Service Provided-created version of the YAML will be distributed to Service Provider applications at design time.

## Goal

The goal of the YAML is to completely describe the processing of VNF events in a way that can be compiled or interpreted by applications across a Service Provider’s infrastructure, so that no additional human configuration or development is required beyond the creation of the YAML itself.

## Relation to the Common Event Format

The Common Event Format described in the VES Event Listener service specification defines the structure of VES events including optional fields that may be provided.

Specific eventNames registered by the YAML (e.g., an InvalidLicense fault), may require that certain fields, which are optional in the Common Event Format, be present when events with that eventName are published. For example, a fault eventName which communicates an ‘InvalidLicense’ condition, may be registered to require that the configured ‘licenseKey’ be provided as a name-value pair in the Common Event Format’s ‘additionalFields’ structure, within the ‘faultFields’ block. Anytime an ‘InvalidLicense’ fault event is detected, designers, applications and microservices across the Service Provider’s infrastructure can count on that name-value pair being present.

The YAML registration may also restrict ranges or enumerations defined in the Common Event Format. For example, eventSeverity is an enumerated string within the Common Event Format with several values ranging from ‘NORMAL’ to ‘CRITICAL’. The YAML registration for a particular eventName may require that it always be sent with eventSeverity set to a single value (e.g., ‘MINOR’), or to a subset of the possible enumerated values allowed by the Common Event Format (e.g., ‘MINOR’ or ‘NORMAL’).

## Relation to Service Design and Creation

Event registration for a VNF (or other event source) is provided to the Service Provider’s Service Creation and Design Environment (e.g., ASDC) as a set of two YAML files consisting of the vendor recommendation YAML and (optionally) the final Service Provider YAML. These YAML files describe all the eventNames that that VNF (or other event source) generates.

Once their events are registered, the Service Creation and Design Environment can then list the registered eventNames (e.g., as a drop down list), for each VNF or other event source (e.g., a service), and enable designers to study the YAML registrations for specific eventNames. The YAMLs provide the information that the designers need to develop and understand policies, work flows and microservices that are triggered by those events. YAML registrations are both human readable and machine readable.

The final Service Provider YAML is a type of Service Design and Creation ‘artifact’, which is distributed to Service Provider applications at design time: notably, to applications involved in the collection and processing of VNF events. It is parsed by those applications so they can automatically support the receipt and processing of VNF events, without the need for any manual, VNF-specific development.

# YAML Files

## YAML Specification Conformance

YAML files should conform to version 1.2 of the YAML specification available at: <http://yaml.org/spec/1.2/spec.html>.

## Filename

YAML file names should conform to the following naming convention:

{AsdcModel}\_{AsdcModelType}\_{v#}.yml

The ‘#’ should be replaced with the current numbered version of the file.

‘ASDC’ is a reference to the Service Provider’s Service Design and Creation environment. The AsdcModelType is an enumeration with several values of which the following three are potentially relevant:

* Service
* Vnf
* VfModule

The AsdcModel is the modelName of the specific modelType whose events are being registered (e.g., the name of the specific VNF or service as it appears in the the Service Design and Creation Environment).

For example:

* vMRF\_Vnf\_v1.yml
* vMRF\_Service\_v1.yml
* vIsbcSsc\_VfModule\_v1.yml

## File Structure

Each eventType is registered as a distinct YAML ‘document’.

YAML files consist of a series of YAML documents delimited by ‘---‘ and ‘…’ for example:

---

# Event Registration for eventName ‘name1’

# details omitted

...

---

# Event Registration for eventName ‘name2’

# details omitted

...

---

# Event Registration for eventName ‘name3’

# details omitted

...

# YAML Syntax and Semantics

YAML registration documents show each relevant VES Common Event Model object and field (i.e., each element) for the eventName being registered, including any extensible fields (e.g., specific name-value pairs).

## Qualifiers

Each object or field name in the eventName being registered is followed by a ‘qualifier’, which consists of a colon and two curly braces, for example:

“objectOrFieldName: { }”

The curly braces contain meta-information about that object or field name (also known as the ‘element’), such as whether it is required to be present, what values it may have, what handling it should trigger, etc…

Semantics have been defined for the following types of meta-information within the curly braces:

### Action

The ‘action’ keyword may be applied to field values or to the event as a whole. The ‘action’ keyword specifies a set of actions that should be taken if a specified trigger occurs. For example, the ‘action’ keyword may specify that a threshold crossing alert (i.e., tca) be generated, and/or that a specific microservice handler be invoked, and/or that a specific named-condition be asserted. In the Rules section of the YAML file, tca’s and microservices may be defined on individual named-conditions or on logical combinations of named-conditions.

The ‘action:’ keyword is followed by five values in square brackets. The first two values communicate the trigger, and the last three values communicate the actions to be taken if that trigger occurs:

1. The first value conveys the trigger level. If the field on which the action is defined reaches or passes through that level, then the trigger fires. If a specific level is not important to the recommended action, the ‘any’ keyword may be used as the first value. (Note: ‘any’ is often used when an action is defined on the ‘event’ structure as a whole).
2. The second value indicates the direction of traversal of the level specified in the first value. The second value may be ‘up’, ‘down’, ‘at’ or ‘any’. ‘any’ is used if the direction of traversal is not important. ‘at’ implies that it traversed (or exactly attained) the trigger level but it doesn’t matter if the traversal was in the up direction or down direction. Note: If ‘up’, ‘down’ or ‘at’ are used, the implication is that the microservices processing the events within the service provider are maintaining state (e.g., to know that a measurement field traversed a trigger level in an ‘up’ direction, the microservice would have to know that the field was previously below the trigger level). When initially implementing support for YAML actions, a service provider may choose to use and interpret these keywords in a simpler way to eliminate the need to handle state. Specifically, they may choose to define and interpret all ‘up’ guidance to mean ‘at the indicated trigger level or greater’, and they may choose to define and interpret all ‘down’ guidance to mean ‘at the indicated trigger level or lower’.
3. The third value optionally names the condition that has been attained when the triggers fires (e.g., ‘invalidLicence’ or ‘capacityExhaustion’). Named-conditions should be expressed in upper camel case with no underscores, hyphens or spaces. In the Rules section of the YAML file, named-conditions may be used to specify tca’s that should be generated and/or microservices that should be invoked. If it is not important to name a condition, then the keyword ‘null’ may be used as the third value.
4. The fourth value recommends a specific microservice (e.g., ‘rebootVm’ or ‘rebuildVnf’) supported by the Service Provider, be invoked if the trigger is attained. Design time processing of the YAML by the service provider can use these directives to automatically establish policies and configure flows that need to be in place to support the recommended runtime behavior.

If a vendor wants to recommend an action, it can either work with the service provider to identify and specify microservices that the service provider support, or, the vendor may simply indicate and recommend a generic microservice function by prefixing ‘RECO-’ in front of the microservice name, which should be expressed in upper camel case with no underscores, hyphens or spaces.

The fourth value may also be set to ‘null’.

1. The fifth value third value indicates a specific threshold crossing alert (i.e., tca) that should be generated if the trigger occurs. This field may be omitted or provided as ‘null’.

Tca’s should be indicated by their eventNames.

When a tca is specified, a YAML registration for that tca eventName should be added to the event registrations within the YAML file.

Examples:

* + event: { action: [ any, any, null, rebootVm ] }

# whenever the above event occurs, the VM should be rebooted

* + fieldname: { action: [ 80, up, null, null, tcaUpEventName ], action: [ 60, down, overcapacity, null ] }

# when the value of fieldname crosses 80 in an up direction, tcaUpEventName

should be published; if the fieldname crosses 60 in a down direction an

‘overCapacity’ named-condition is asserted.

### Array

The ‘array’ keyword indicates that the element is an array; ‘array:’ is following by square brackets which contain the elements of the array. Note that unlike JSON itself, the YAML registration will explicitly declare the array elements and will not communicate them anonymously.

Examples:

* + element: { array: [

firstArrayElement: { },

secondArrayElement: { }

] }

### Default

The ‘default’ keyword specifies a default field value. Note: the default value must be within the range or enumeration of acceptable values.

Examples:

* + fieldname: { range: [ 1, unbounded ], default: 5 }
  + fieldname: { value: [ red, white, blue ], default: blue }

### HeartbeatAction

The ‘heartbeatAction’ keyword is provided on the ‘event’ objectName for heartbeat events only. It provides design time guidance to the service provider’s heartbeat processing applications (i.e., their watchdog timers). The syntax and semantics of the ‘heartbeatAction’ keyword are similar to the ‘action’ keyword except the trigger is specified by the first field only instead of the first two fields. When the ‘heartbeatAction’ keyword is indicated, the first field is an integer indicating the number of successively missed heartbeat events. Should that trigger occur, the remaining fields have the same order, meaning and optionality as those described for the ‘action’ keyword.

Examples:

* + event: { heartbeatAction: [ 3, vnfDown, RECO-rebootVnf, tcaEventName ] }

# whenever the above event occurs, a vnfDown condition is asserted and the vnf should be rebooted, plus the indicated tca should be generated.

### Presence

The ‘presence’ keyword may be defined as ‘required’ or ‘optional’. If not provided, the element is assumed to be ‘optional’.

Examples

* + element: { presence: required } # element must be present
  + element: { presence: optional } # element is optional
  + element: { value: blue } # by omitting a presence definition, the

element is assumed to be optional

### Range

The ‘range’ keyword applies to fields (i.e., simpleTypes); indicates the value of the field is a number within a specified range of values from low to high (inclusive of the indicated values). . ‘range:’ is followed by two parameters in square brackets:

* + the first parameter conveys the minimum value
  + the second parameter conveys the maximum value or ‘unbounded’

The keyword ‘unbounded’ is supported to convey an unbounded upper limit. Note that the range cannot override any restrictions defined in the VES Common Event Format.

Examples:

* + fieldname: { range: [ 1, unbounded ] }
  + fieldname: { range: [ 0, 3.14 ] }

### Structure

The ‘structure’ keyword indicates that the element is a complexType (i.e., an object) and is followed by curly braces containing that object.

Example:

* objectName: { structure: {

element1: { },

element2: { },

anotherObject: { structure: {

element3: { },

element4: { }

} }

} }

### Value

The ‘value’ keyword applies to fields (i.e., simpleTypes); indicates a single value or an enumeration of possible values. If not provided, it is assumed the value will be determined at runtime. Note that the declared value cannot be inconsistent with restrictions defined in the VES Common Event Format (e.g., it cannot add an enumerated value to an enumeration defined in the Common Event Format, but it can subset the defined enumerations in the Common Event Format).

Values that are strings containing spaces should always be indicated in single quotes.

Examples:

* + fieldname: { value: x } # the value is ‘x’
  + fieldname: { value: [ x, y, z ] } # the value is either ‘x’, ‘y’, or ‘z’
  + fieldname: { presence: required } # the value will be provided at runtime
  + fieldname: { value: ‘error state’ } # the value is the string within the single quotes

## Rules

### Rules Document

After all events have been defined, the YAML file may conclude with a final YAML document delimited by ‘---‘ and ‘…’, which defines rules based on the named ‘conditions’ asserted in action qualifiers in the preceding event definitions. For example:

---

# Event Registration for eventName ‘name1’

event: {presence: required, action: [any, any, A, null], structure: {

# details omitted

}}

...

---

# Event Registration for eventName ‘name2’

event: {presence: required, structure: {

commonEventHeader: {presence: required, structure: {

# details omitted

}}

measurementsForVfScalingFields: {presence: required, structure: {

cpuUsageArray: {presence: required, array: {

cpuUsage: {presence: required, structure: {

cpuIdentifier: {presence: required},

percentUsage: {presence: required, action: [90, up, B, null]}

}}

}},

# details omitted

}}

}}

...

---

# Rules

rules: [

# defined based on conditions ‘A’ and ‘B’ - details omitted

]

...

### Rules Syntax and Semantics

The YAML ‘rules’ document begins with the keyword ‘rules’ followed by a colon and square brackets. Each rule is then defined within the square brackets. Commas are used to separate rules.

Each rule is expressed as follows:

rule: {

trigger: *logical expression in terms of conditions*,

microservices: [ *microservice1, microservice2, microservice3…* ]

alerts: [tcaE*ventName1, tcaEventName2, tcaEventName3…* ],

}

Notes:

* All referenced tcaEventNames should be defined within the YAML.
* For information about microservices, see section 3.1.1 bullet number 4.
* At least one microservice or alert should be specified, and both microservices and alerts may be specified.

### Simple Triggers

The trigger is based on the named ‘conditions’ asserted in the action qualifiers within the event definitions earlier in the YAML file. The following logical operators are supported:

* &: which is a logical AND
* ||, which is a logical OR

In addition parentheses may be used to group expressions.

Example logical expression:

(A & B) || (C & D)

Where A, B, C and D are named conditions expressed earlier in the YAML file.

Example rules definition:

rules: [

rule: {

trigger: A,

alerts: [tcaEventName1],

microservices: [rebootVm]

},

rule: {

trigger: B || (C & D),

microservices: [scaleOut]

}

]

Note: when microservices are defined in terms of multiple event conditions, the designer should take care to consider whether the target of the microservice is clear (e.g., which VNF or VM instance to perform the action on). Future versions of this document may provide more clarity.

### Time Based Qualifiers

Time based rules may be established by following any named condition with a colon and curly braces. The time based rule is placed in the curly braces as follows:

trigger: B:{3 times in 300 seconds}

This means that if condition B occurs 3 (or more) times in 300 seconds (e.g., 5 minutes), the trigger fires.

More complex triggers can be created as follows:

trigger: B:{3 times in 300 seconds} || (C & D:{2 times in 600 seconds}),

This means that the trigger fires if condition B occurs 3 (or more) times in 5 minutes, OR, if condition D occurs 2 (or more) times in 10 minutes AND condition C is in effect.

# YAML Examples

An example YAML file is provided below which registers events for the vMRF VNF.

---

# registration for Fault\_vMRF\_InvalidLicense

event: {presence: required, action: [any, any, invalidLicense, RECO-renewLicence], structure: {

commonEventHeader: {presence: required, structure: {

domain: {presence: required, value: fault},

eventName: {presence: required, value: Fault\_vMRF\_InvalidLicense},

eventId: {presence: required},

nfType: {presence: required, value: mrfx},

priority: {presence: required, value: High},

reportingEntityName: {presence: required},

sequence: {presence: required},

sourceName: {presence: required},

startEpochMicrosec: {presence: required},

lastEpochMicrosec: {presence: required},

version: {presence: required, value: 3.0}

}},

faultFields: {presence: required, structure: {

faultFieldsVersion: {presence: required, value: 1.2},

alarmCondition: {presence: required, value: ‘Invalid license key’},

eventSourceType: {presence: required, value: virtualNetworkFunction},

specificProblem: {presence: required, value: ‘The node license key is invalid’},

eventSeverity: {presence: required, value: CRITICAL},

vfStatus: {presence: required, value: Active},

alarmAdditionalInformation: {presence: required, array: [

field: {presence: required, structure: {

name: {presence: required, value: license\_key},

value: {presence: required}

}}

]}

}}

}}

...

---

# registration for Heartbeat\_vMRF

event: {presence: required, heartbeatAction: [3, vnfDown, RECO-rebuildVnf,

Tca\_vMRF\_vnfDown], structure: {

commonEventHeader: {presence: required, structure: {

domain: {presence: required, value: fault},

eventName: {presence: required, value: Heartbeat\_vMRF },

eventId: {presence: required},

nfType: {presence: required, value: mrfx},

priority: {presence: required, value: High},

reportingEntityName: {presence: required},

sequence: {presence: required},

sourceName: {presence: required},

startEpochMicrosec: {presence: required},

lastEpochMicrosec: {presence: required},

version: {presence: required, value: 3.0}

}},

heartbeatFields: { presence: optional, structure: {

heartbeatFieldsVersion: {presence: required, value: 1.0},

heartbeatInterval: {presence: required, range: [ 0, 600 ], default: 60 }

}}

}}

...

---

# registration for Mfvs\_vMRF

event: {presence: required, structure: {

commonEventHeader: {presence: required, structure: {

domain: {presence: required, value: measurementsForVfScaling},

eventName: {presence: required, value: Mfvs\_vMRF},

eventId: {presence: required},

nfType: {presence: required, value: mrfx},

priority: {presence: required, value: Normal},

reportingEntityName: {presence: required},

sequence: {presence: required},

sourceName: {presence: required},

startEpochMicrosec: {presence: required},

lastEpochMicrosec: {presence: required},

version: {presence: required, value: 3.0}

}},

measurementsForVfScalingFields: {presence: required, structure: {

measurementsForVfSclaingFieldsVersion: {presence: required, value: 2.0},

measurementInterval: {presence: required, range: [ 60, 1200 ], default: 180 },

concurrentSessions: {presence: required},

cpuUsageArray: {presence: required, array: {

cpuUsage: {presence: required, structure: {

cpuIdentifier: {presence: required},

percentUsage: {presence: required, range: [ 0, 100 ], action: [

90, up, CpuUsageHigh, RECO-scaleOut,

Tca\_vMRF\_HighCpuUsage ],

action: [25, down, CpuUsageLow, RECO-scaleIn,

Tca\_vMRF\_LowCpuUsage

]}

}}

}},

memoryUsageArray: {presence: required, array: {

memoryUsage: {presence: required, structure: {

vmIdentifier: {presence: required},

memoryFree: {presence: required, range: [ 0, 100 ], action: [

100, down, FreeMemLow, RECO-scaleOut,

Tca\_vMRF\_LowFreeMemory ], action: [1000, up, FreeMemHigh,

RECO-scaleIn, Tca\_vMRF\_HighFreeMemory

]},

memoryUsed: {presence: required}

}}

}},

numberOfMediaPortsInUse: {presence: required, range: [ 1, 300 ] },

additionalMeasurements: {presence: required, array: [

measurementGroup: {presence: required, structure: {

name: {presence: required, value: licenseUsage},

measurements: {presence: required, array: [

field: {presence: required, structure: {

name: {presence: required, value: [ G711AudioPort,

G729AudioPort, G722AudioPort, AMRAudioPort,

AMRWBAudioPort, OpusAudioPort, H263VideoPort,

H264NonHCVideoPort, H264HCVideoPort, MPEG4VideoPort,

NP8NonHCVideoPort, VP8HCVideoPort, PLC, NR, NG, NLD,

G711FaxPort, T38FaxPort, RFactor, T140TextPort ] },

value: {presence: required}

}}

]}

}}

]}

}}

}}

...

---

# registration for Syslog\_vMRF

event: {presence: required, structure: {

commonEventHeader: {presence: required, structure: {

domain: {presence: required, value: syslog},

eventName: {presence: required, value: Syslog\_vMRF},

eventId: {presence: required},

nfType: {presence: required, value: mrfx},

priority: {presence: required, value: Normal},

reportingEntityName: {presence: required},

sequence: {presence: required},

sourceName: {presence: required},

startEpochMicrosec: {presence: required},

lastEpochMicrosec: {presence: required},

version: {presence: required, value: 3.0}

}},

syslogFields: {presence: required, structure: {

eventSourceHost: {presence: required},

eventSourceType: {presence: required, value: virtualNetworkFunction},

syslogFacility: {presence: required, range: [0, 23]},

syslogFieldsVersion: {presence: required, value: 3.0},

syslogMsg: {presence: required},

syslogPri: {presence: required, range: [0, 192]},

syslogProc: {presence: required, range: [0, 65536]},

syslogSData: {},

syslogSdId: {},

syslogSev: {presence: required},

syslogTag: {presence: required, value: vMRF},

syslogVer: {presence: required, value 0}

}}

}}

...

---

# Rules

Rules: [

rule: {

trigger: CpuUsageHigh && FreeMemLow,

microservices: [scaleOut] # Note: this presumes there is a scaleOut microservice

alerts: [Tca\_vMRF\_OutOfResources] # Note: this TCA should be defined in the YAML

}

rule: {

trigger: CpuUsageLow && FreeMemHigh,

microservices: [scaleIn] # Note: this presumes there is a scaleIn microservice

}

]

...

# Appendix: Historical Change Log

For the latest changes, see the Change Block just before the Table of Contents.

| Date | Revision | Description |
| --- | --- | --- |
| 3/15/2017 | 1.0 | This is the initial release of the VES Event Registration document. |
| 3/22/2017 | 1.1 | * Changed the ‘alert’ qualifier to ‘action’ and added support for conditions that will trigger rules. * Formatted the document with more sections and subsections. * Defined the syntax and semantics for condition based rules. * Fixed the YAML examples. |
| 3/27/2017 | 1.2 | * Clarified the audience of the document and the expectations for vendors. * Changed the order of fields in the action keyword. * Updated the YAML examples. * Wordsmithed throughout. |
| 3/31/2017 | 1.3 | * Generalized the descriptions from an ASDC, ECOMP and AT&T-specific interaction with a VNF vendor, to a generic Service Provider interaction with a VNF vendor. * Wordsmithed throughout. * Added a ‘default’ qualifier * Fixed syntax and semantic inconsistencies in the Rules section * Brought all examples into compliance with v5.0 * Added a heartbeat example * Modified the mfvs example * Modified the syslog example * Added two complex rules |