Velocity Abstract Resource Conditions Language Reference Guide

DOC10678 (KB Document ID)

Contents

Overview

This document describes the language that you use to specify the ***conditions*** that describe an abstract resource.

Abstract resources: You use the language to specify the conditions that describe an ***abstract*** resource. Example conditions are: “port must support 10MBPS” or “resource has at least 3 ports" or "resource must be based on the Acme router template”.

Note: The conditional statement ANY is allowed only for abstract ports and not allowed for abstract resources (The ANY condition on abstract resources causes performance problems for the Velocity resolve algorithm).

Logical topologies: To prepare to reserve resources, you define a logical topology — the topology is logically defined by conditions that must be met during the reserved period. At the appropriate time, Velocity finds and connects the concrete resources and ports that meet the conditions (that is, Velocity ***resolves*** the logical topology to a concrete topology). A logical topology that makes use of abstract resources is called an abstract topology.

During the process of resolving the topology, Velocity resolves abstract resources and also maps the logical connections that are defined in the topology into concrete connections through switches and their ports. The resulting concrete resources and concrete connections make up the resolved ***concrete*** topology and are used for the duration of the reservation.

Resolving the abstract resources and ports to physical resources and connections for a reservation: While configuring a reservation, Velocity considers both the conditions specified for the abstract resources and the connectivity restrictions and resource costs. If Velocity cannot resolve an abstract resource and make all of the connections, then the reservation fails.

Part 1: Velocity compares the concrete resources in the inventory against the specified conditions to determine whether to include a particular resource. If a resource meets the conditions, then Velocity uses the resource in the reservation – this (along with the cost minimization described in a moment) is called ***resolving*** the logical topology to a concrete topology.

-
  - For recurring reservations, an abstract topology is resolved to a concrete topology when a reservation instance starts.
  - For non-recurring reservations, the topology is resolved immediately after the reservation is scheduled.

Part 2: Velocity calculates resource cost based on other reservations that affect the resource. The cost is higher for resources for which conflicts are more likely. Velocity tries to resolve abstract resources so that the sum of all costs for the abstract topology is a minimum. The solution does not guarantee the best cost but provides a reasonable level of optimization.

A ***concrete*** topology refers to physical resources by their UUID (universal unique identifiers). A logical topology can also be considered concrete if it does not contain abstract resources.

Condition Language

Examples

```
// has 10 or more ports of any kind
PORTS >= 10

// either the 'modelNumber' property is less than 10 or the 'series' property is equal to 'xxx-xx-xxx'
int[modelNumber] < 10 OR [series] = "xxx-xx-xxx"

// query of a resource from series 'Router 123' with at least four 10Gb Ethernet ports
[Model] = 'Router 123' AND PORTS([Port Type] = 'Ethernet' AND int[Port Speed] = 10000) >= 4

// query of a resource from series 'Router 123' with at least four 10Gb Ethernet ports that are online
[Model] = 'Router 123' AND PORTS([Port Type] = 'Ethernet' AND int[Port Speed] = 10000) >= 4 AND status [online]

// query of a resource inherited from Firewall template that has 'Model' matching the wildcard string
// this will match: 'FOO 155-X', 'FOO 355768-C', 'FOO 555'
template[Firewall] AND [Model] LIKE 'FOO _55%'

// query of a good port with optical return loss less than 38.4 dB
boolean[good] = true AND decimal[opticalReturnLoss] < 38.4
```

Abstract Resource in TBML

A resource in TBML is marked as abstract using the `com.fnfr.rm.condition extension.`

```
<resource guid="a0ea02d7-52d8-40b5-94f5-c41afadf3c74" id="resource_0" type="server">
   ...
   <extensions type="com.fnfr.rm.condition" xs:namespace="com.fnfr.rm.condition">
      <!-- abstract resource conditions -->
   </extensions>
   ...
</resource>
```

If a resource in TBML has the `com.fnfr.rm.condition` block, it is considered abstract, even if a GUID is also present. The block must not be empty.

Use the `any` condition to mark a resource as abstract without specifying particular conditions.

We strongly advise against using `any` condition for resources, however. The `any` condition is meant to be used primarily for ports.

Data Types

The following table lists the conditional operations supported by the various data types. Each operation takes 2 arguments and returns a Boolean value.

Data Type	Operations
Boolean	=
Integer	=, <, >, <=, >=
Decimal	=, <, >, <=, >=
String	=, like*

* `like` is similar to SQL's `LIKE` condition.

Keywords

Keywords can be written either in lowercase ("and") or in uppercase ("AND"), but mixed case ("And") is **not** supported. Here is the list of keywords of Velocity condition language:

integer int boolean bool decimal string

or and not like any template ports

true and *false* might seem like keywords, but they are actually literals. You can only write them in lowercase.

Condition syntax and semantics for resources

Each condition is treated like a predicate that can return `true` or `false`.

Any

```
<extensions type="com.fnfr.rm.condition" xs:namespace="com.fnfr.rm.condition"> any
</extensions>
```

Always returns `true`. The '`any`' condition is used to define an abstract port when you do not require its properties/template/etc. For example, you want a resource with certain properties and one port, but do not care which

of the multiple ports of the resource will be used. Specify the desired conditions for the resource and specify the 'any' condition for its port.

Note: The 'any' condition is only allowed for abstract ports, not abstract resources.

Property

```
integer [propertyName] < value
```

Returns `true` if a property with the specified name exists, is of the specified type, and the comparison with the specified value is also `true`.

Types: `integer/int, boolean/bool, decimal, string`. Type may be skipped (`string` type is the default, like this: `[Hostname]="switch"`)

Operations: `=, >, <, =>, <=, %` (aliases: `eq, gt, lt, ge, le, like`). `eq` is valid for each type, `like` is for `string` type only, the others are for `integer` and `decimal` types.

Boolean type values are `true` and `false`. String type values must be enclosed in single or double quotes. Integer and decimal value do not require quotes.

Template

```
template [templateName]
```

Returns `true` if the resource is derived from the specified template (directly or indirectly). Otherwise returns `false`, including the case where the specified template is not found. For example, `template [Cisco Router]`

Status

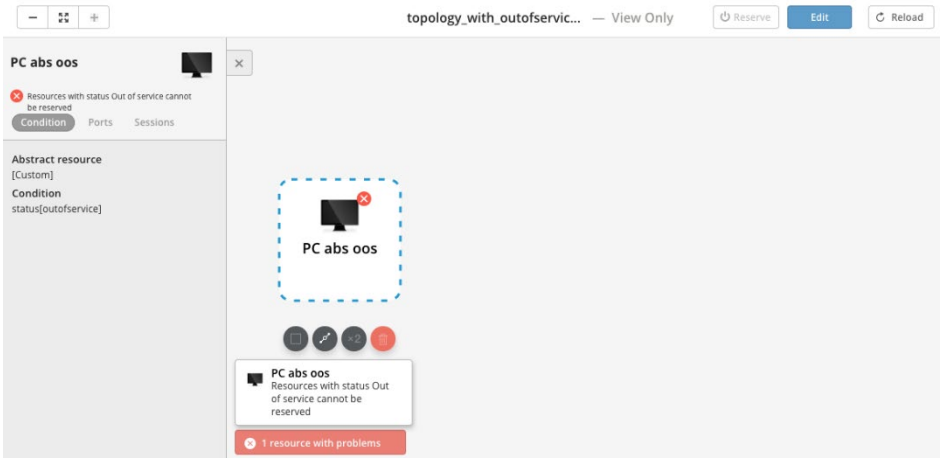
```
status [online]
```

Returns `true` if the resource has the specified status value. Otherwise returns `false`. Available status values for this condition:

- status [online]
- status [offline]
- status [outofservice]
- status [unknown]

Resources with condition "status [outofservice]"

will be marked with special error message:



Ports

`PORTS(<sub-condition>) >= integerValue`

Calculates the number of ports that conform to the specified sub-condition (which may be empty, in which case the `any` sub-condition is assumed).

Returns `true` if the number of such ports satisfies the specified comparison.

Operations: `=, >, <, =>, <=` (aliases: `eq, gt, lt, ge, le`) are supported. Any valid condition expression is allowed to specify conditions for ports (except for the `ports` condition itself).

`(..)` block is optional. For example, to ensure that a resource has at least 3 PORTS, use: PORTS `>= 3`

Parent

`PARENT(<sub-condition>)`

Returns true if the resource is directly nested under the parent resource that conforms to the specified condition. For example, *parent(template[STC])*.

Any valid condition expression is allowed to specify conditions for parent resource (except for the *parent* condition itself).

Ancestor

`ANCESTOR(<sub-condition>)`

Returns true if the resource is nested (directly or indirectly) under an ancestor resource that conforms to the specified condition. For example, *ancestor(template[STC])*.

Any valid condition expression is allowed to specify conditions for ancestor resource (except for the *ancestor* condition itself).

Not

```
not <sub-condition>
```

Returns `true` if the specified sub-condition returns `false`.

And

```
<sub-condition1> and <sub-condition2> and ...
```

Returns `true` if all of the sub-conditions return `true`.

Or

```
<sub-condition1> or <sub-condition2> ...
```

Returns `true` if any of the sub-conditions return `true`.

Condition syntax and semantics for links

In addition to abstract resource and abstract port conditions, Velocity also supports abstract link conditions. Users can define an abstract link condition to a layer 1 connection between devices in a topology to constrain the resource and port selections in an abstract topology. All logical operations for link conditions work the same way as they do for resources. In addition, there are some keywords that apply only to links.

With the exception of the "WithEdges" condition, link conditions apply only to cables and ports in the route between resources (intermediate elements like cables, patch panels, and layer 1 switches), exclusive of the endpoint's (resource) ports.

Note: Link conditions apply only to logical links between endpoints with L1 connection type. Link conditions for L2 links (simple VLAN connections, VLANs, MultiVLANs) are not supported.

Port

```
port[propertyName] = "propertValue"
```

Returns true if string strictly matches on all intermediate ports.

Cable

```
cable[propertyName] = "propertValue"
```

Returns true if string strictly matches on all intermediate cables.

Sum

```
sum([propertyName]) = value
```

Note: The Sum condition supports only INTEGER or DECIMAL property types.

Returns true if the result of a numeric evaluation on the sum of all intermediate ports and cables matches value.

Operations: `=, >, <, =>, <=` (aliases: `eq, gt, lt, ge, le`) are supported.

WithEdges

The following syntax applies the condition to all ports on the route (including endpoint ports). It can be used for the Sum condition or Port condition.

```
withEdges <condition>
```

`withEdges port[Property Name] = "Property Value"`	means strict string matches on all ports (intermediate and endpoint ports)
`withEdges sum(port[Property Name]) >,<,= value`	means numeric evaluation on all ports (intermediate and endpoint ports)

Examples

`withEdges port[Port Type] = "Optical"`	intermediate and endpoint ports should have "Port Type" port property with "Optical" value
`withEdges sum(port[Attenuation]) < 25`	intermediate and endpoint ports should have numeric "Attenuation" port property and the sum should be less than 25

Link Conditions Syntax

`[Property Name] = "propertyValue"`	means strict string matches on all intermediate ports and cables
`port[Property Name] = "propertyValue"`	means strict string matches on all intermediate ports
`withEdges port[Property Name] = "propertyValue"`	means strict string matches on all ports including endpoints
`cable[Property Name] = "propertyValue"`	means strict string matches on all cables
`sum([Property Name]) >,<,= value`	means numeric evaluation on all intermediate ports and cables
`sum(port[Property Name]) >,<,= value`	means numeric evaluation on all intermediate ports
`withEdges sum(port[Property Name]) >,<,= value`	means numeric evaluation on all ports including endpoints
`sum(cable[Property Name]) >,<,= value`	means numeric evaluation on all cables

Examples of link conditions

Total length is less than 50m and Attenuation of each hop is less than 25 dB

```
sum(cable[Length]) < 50 AND decimal cable[Attenuation] < 25
```

Only optical cables in Single mode

```
template[Optical Fiber] AND cable[Mode] = 'Single'
```