

# Hitachi Ops Center Analyzer Detail View

---

## Query Language User Guide

This document describes how to use Ops Center Analyzer detail view Query language.

© 2016, 2020 Hitachi, Ltd. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including copying and recording, or stored in a database or retrieval system for commercial purposes without the express written permission of Hitachi, Ltd., or Hitachi Vantara Corporation (collectively "Hitachi"). Licensee may make copies of the Materials provided that any such copy is: (i) created as an essential step in utilization of the Software as licensed and is used in no other manner; or (ii) used for archival purposes. Licensee may not make any other copies of the Materials. "Materials" mean text, data, photographs, graphics, audio, video and documents.

Hitachi reserves the right to make changes to this Material at any time without notice and assumes no responsibility for its use. The Materials contain the most current information available at the time of publication.

Some of the features described in the Materials might not be currently available. Refer to the most recent product announcement for information about feature and product availability, or contact Hitachi Vantara Corporation at [https://support.hitachivantara.com/en\\_us/contactus](https://support.hitachivantara.com/en_us/contactus).

**Notice:** Hitachi products and services can be ordered only under the terms and conditions of the applicable Hitachi agreements. The use of Hitachi products is governed by the terms of your agreements with Hitachi Vantara Corporation.

By using this software, you agree that you are responsible for:

1. Acquiring the relevant consents as may be required under local privacy laws or otherwise from authorized employees and other individuals; and
2. Verifying that your data continues to be held, retrieved, deleted, or otherwise processed in accordance with relevant laws.

**Notice on Export Controls.** The technical data and technology inherent in this Document may be subject to U.S. export control laws, including the U.S. Export Administration Act and its associated regulations, and may be subject to export or import regulations in other countries. Reader agrees to comply strictly with all such regulations and acknowledges that Reader has the responsibility to obtain licenses to export, re-export, or import the Document and any Compliant Products.

Hitachi is a registered trademark of Hitachi, Ltd., in the United States and other countries.

AIX, AS/400e, DB2, Domino, DS6000, DS8000, Enterprise Storage Server, eServer, FICON, FlashCopy, IBM, Lotus, MVS, OS/390, PowerPC, RS/6000, S/390, System z9, System z10, Tivoli, z/OS, z9, z10, z13, z/VM, and z/VSE are registered trademarks or trademarks of International Business Machines Corporation.

Active Directory, ActiveX, Bing, Excel, Hyper-V, Internet Explorer, the Internet Explorer logo, Microsoft, the Microsoft Corporate Logo, MS-DOS, Outlook, PowerPoint, SharePoint, Silverlight, SmartScreen, SQL Server, Visual Basic, Visual C++, Visual Studio, Windows, the Windows logo, Windows Azure, Windows PowerShell, Windows Server, the Windows start button, and Windows Vista are registered trademarks or trademarks of Microsoft Corporation. Microsoft product screen shots are reprinted with permission from Microsoft Corporation.

All other trademarks, service marks, and company names in this document or website are properties of their respective owners.

**EXPORT CONTROLS** - Licensee will comply fully with all applicable export laws and regulations of the United States and other countries, and Licensee shall not export, or allow the export or re-export of, the Software, API or Materials in violation of any such laws or regulations. By downloading or using the Software, API, or Materials, Licensee agrees to the foregoing and represents and warrants that Licensee is not located in, under the control of, or a national or resident of any embargoed or restricted country.

# Contents

<b>1</b>	<b>Preface .....</b>	<b>5</b>
1.1	Product version.....	5
1.2	Intended audience .....	5
1.3	Accessing product documentation.....	5
1.4	Getting help .....	5
1.5	Comments.....	5
<b>2</b>	<b>Analyzer detail view query language .....</b>	<b>6</b>
<b>3</b>	<b>Query language specification.....</b>	<b>7</b>
3.1	Terminology .....	7
3.1.1	Overview.....	8
3.1.2	Query structure .....	8
3.1.3	Resource filters.....	9
3.1.4	Attribute filters .....	9
<b>4</b>	<b>Query language BNF .....</b>	<b>20</b>
4.1	Query statement.....	20
4.2	Scalar filter .....	20
4.3	Timeseries filter .....	20
4.3.1	Timeseries filter expression .....	21
4.4	Subsequence filter .....	21
4.5	Non-terminals.....	21
4.6	Limitations .....	21
<b>5</b>	<b>Advanced data processing functions .....</b>	<b>22</b>
5.1	Query options.....	22
5.1.1	Query global options.....	22
5.1.2	Query filter options.....	23
5.2	Interval Rollup.....	24
5.2.1	Syntax .....	25
5.2.2	Supported operations.....	26

5.2.3	Use case 1 .....	26
5.2.4	Use case 2 .....	31
5.3	Derived attributes .....	32
5.3.1	Syntax .....	32
5.3.2	Supported operations.....	34
5.3.3	Use case .....	34
5.4	Resource Rollup.....	40
5.4.1	Syntax .....	41
5.4.2	Supported operations.....	45
5.4.3	Use case .....	45
5.5	preProc .....	48
5.5.1	Syntax .....	48
5.5.2	Use case .....	48
6	External scripts.....	50
6.1	Supported scripting languages.....	50
6.1.1	R.....	50
6.1.2	Groovy.....	51
6.1.3	Python .....	51
6.2	Script filter .....	51
6.2.1	Regular external scripts.....	53
6.2.2	Rollup external scripts.....	63
6.2.3	Group external scripts.....	70
7	Advanced features.....	80
7.1	Query Join.....	80
7.1.1	Syntax .....	80
7.1.2	Use case .....	82
7.2	Synthetic attribute .....	83
7.2.1	Syntax .....	83
7.2.2	Use case .....	84
8	Appendix.....	85

# 1 Preface

This document describes how to use Hitachi Ops Center Analyzer detail view Query language.

## 1.1 Product version

This document revision applies to Hitachi Ops Center Analyzer detail view version 10.2 or later.

## 1.2 Intended audience

This document is intended for Infrastructure Administrators who manage complex data centres.

## 1.3 Accessing product documentation

Product documentation is available on Hitachi Vantara Support Connect: <https://knowledge.hitachivantara.com/Documents>. Check this site for the most current documentation, including important updates that may have been made after the release of the product.

## 1.4 Getting help

[Hitachi Vantara Support Connect](https://support.hitachivantara.com/en_us/contact-us.html) is the destination for technical support of products and solutions sold by Hitachi Vantara. To contact technical support, log on to Hitachi Vantara Support Connect for contact information: [https://support.hitachivantara.com/en\\_us/contact-us.html](https://support.hitachivantara.com/en_us/contact-us.html).

[Hitachi Vantara Community](https://community.hitachivantara.com) is a global online community for Hitachi Vantarac customers, partners, independent software vendors, employees, and prospects. It is the destination to get answers, discover insights, and make connections. **Join the conversation today!** Go to [community.hitachivantara.com](https://community.hitachivantara.com), register, and complete your profile.

## 1.5 Comments

Please send us your comments on this document to [doc.comments@hitachivantara.com](mailto:doc.comments@hitachivantara.com). Include the document title and number, including the revision level (for example, -07), and refer to specific sections and paragraphs whenever possible. All comments become the property of Hitachi Vantara Corporation.

**Thank you!**

## 2 Analyzer detail view query language

The Analyzer detail view query language (referred to as MQL in this document) is a regex-based terse yet expressive query language used to retrieve and filter data stored in Analyzer detail view database.

MQL allows complex analysis on this data in real-time with constant run-time.

MQL syntax makes it possible to traverse relations, identify patterns in data, and provides a mechanism to establish a correlation.

MQL query consists of the following parameters:

- A single line query
- Start time
- End time

You can perform advanced operations like interval rollup, resource rollup etc. using MQL by referring to this guide. For example, you can use the interval rollup operation to aggregate data from a lower interval to a higher interval. To perform advanced operations using MQL, log on to Analyzer detail view, click **Reports > Report Builder** and then click **Create Using MQL** option.

**Note:** You can also build queries through Query Builder option in the Analyzer detail view UI. However, advanced MQL features are not supported in the Query Builder. For more information, refer to the **Creating custom reports using Query Builder** section in the *Hitachi Ops Center Analyzer Detail View Help*.

# 3 Query language specification

## 3.1 Terminology

- **Resource:** An entity for which scalar and timeseries data exists in the Analyzer detail view database. Resources may represent real world entities such as a Host, CPU, Storage Volume, Storage System, NIC Card, and Virtual Machine.
- **Attribute:** A property that is attached to a resource which provides more information about it. Two types of attributes are as follows:
  - **Scalar attributes** that might not change frequently. For example, host name or IP address of a host, size of a storage volume.
  - **Timeseries attributes** store timeseries numerical data for a resource. For example, CPU usage of a host, number of IOPS occurring on a storage volume.
- **Relation:** A resource in real world typically does not operate independently. It is either dependent on another resource or contains resources within or under itself. For example, a VM is dependent on the Host it runs. Vice-versa the Host contains a number of VMs which are running on it. Two resources are considered related if one resource depends on another resource or one resource contains the other resource.
- **Resource Definition:** Defines how the resource is modeled. The definition consists of the following:
  - **Type:** An unique identifier for resource definition within a given dataset. For example, LHost can represent a Linux Host, LHostCPU can represent the CPUs in a host.
  - **Scalar Attributes:** IDs of scalar attribute definition for which data can be stored in resources of this type, for example, name, ipAddress, capacity.
  - **Timeseries Attributes:** IDs of timeseries attribute definition for which data can be stored in resources of this type, for example, cpuUsage, memUsage, readIOPS.
  - **Relations:** List of resource types which can be related to resource belonging to this definition, for example, LHost is related with LHostCPU.
- **Attribute Definition:** Defines how the attribute is modeled. The definition consists of the following:
  - **ID:** An unique identifier for attribute definition within a given dataset, for example, name, ipAddress, cpuUsage
  - **Name:** Display name for the attribute definition
  - **Type:** Attribute definition type. Valid values are scalar and timeseries.
  - **Unit:** Specifies the unit in which the value is stored, for example, Percent, MBps, KBps.

### 3.1.1 Overview

MQL query consists of a series of resource filters separated by the path separator '/'. Each resource filter progressively narrows down from the entire set of resources stored inside the Analyzer detail view database to a smaller set based on the filter criteria.

The very first filter operates on the entire set of resources stored inside the Analyzer detail view database. The successive filters operate on the related resources of the resources matched in the upper level resource filter. Therefore, the very first filter can start by filtering any resource type.

### 3.1.2 Query structure

`resource_type[filter].../related_resource_type[filter]/...`

- Must contain at least one level
- No limit on number of levels
- Filters are optional

Let's look at few query examples:

- **Example 1: Single resource type without any filter**

`LHost`

- Consists of a single resource type with no filters.
- Returns all the resources of *LHost* type.

- **Example 2: Resource type followed by a scalar filter**

`LHost[=name rx node]`

- Consists of resource type followed by a scalar filter.
- Returns all the resources of *LHost* type, having scalar attribute *name* whose value matches the regular expression *node*.

- **Example 3: Resource type followed by a timeseries filter**

`LHost[@L_user rx b .*]`

- Consists of resource type followed by a timeseries filter.
- Returns all the resources *LHost* type, having the timeseries attribute *L\_user* whose value matches the regular expression '*.\**'.

- **Example 4: Two resource types followed by scalar attribute for each resource type**

`LHost[=name rx node]/LHostDisk[=name rx sda]`

- Consists of two resource filters (*LHost* and *LHostDisk*). In the second filter, resource *LHostDisk* is mentioned because *LHost* is related to *LHostDisk*.



- Returns all resources of *LHost* type with name matching the regular expression *node*, and their related resources of *LHostDisk* type name matching the regular expression *sda*.
- **Example 5: One resource type followed by one scalar and one timeseries filter**  
`LHost[=name rx node]&[@L_user rx b [U50-U100]{5,}]`
  - Combines a scalar and a timeseries filter.
  - Returns all resources of *LHost* type with attribute *name* matching regular expression *node*, and timeseries filter on attribute *L\_user* matching the regular expression *[U50-U100]{5,}*.
  - The timeseries data returned are those points where *L\_user* was between 50 and 100 for 5 consecutive data points.

### 3.1.3 Resource filters

A resource filter consists of a resource type and any number of attribute filters. Each attribute filter is enclosed between the big angle brackets, for example, `r[f1][f2]`. By default, logical *OR* is applied between two filters. A logical *AND* can be applied between two filters using the `&` operator, for example, `r [f1] & [f2]`.

Logical *OR* and *AND* operations can be applied between resource filters separated by path separator. By default, the operation is logical *OR*. For example:

- `r1[f1]/r2[f2]`. Results of `r1[f1]` are included irrespective of whether a match occurred at `r2[f2]`
- `r1[f1]/&r2[f2]`. Only those results of `r1[f1]` are included that have a match at `r2[f2]`

A special character *\** (*asterisk*) can be prefixed to any resource filter. This implies that the results of this resource filter will be suppressed in the output results. This is useful in situations where results from a subsequent resource filter are required but this resource filter is required as a passthrough, for example, `*r1[f1]/r2[f2]`.

### 3.1.4 Attribute filters

#### 3.1.4.1 Scalar filters

A scalar filter is used to filter resources based on their scalar attribute values using string regular expression. Any regular expression constructed as per JDK 1.7 specifications (<http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html>) is supported. The filter tries to find a subsequence using the specified regular expression as opposed to a complete match.

- **Scalar filter structure:**

```
<resource_type>[=<attribute_id> rx regex]
```

**Example:** Get all hosts that have attribute *name* containing *Host*

```
h[=name rx Host]
```

Multiple scalar filters on different attributes can be specified using logical operators 'OR' and 'AND'.

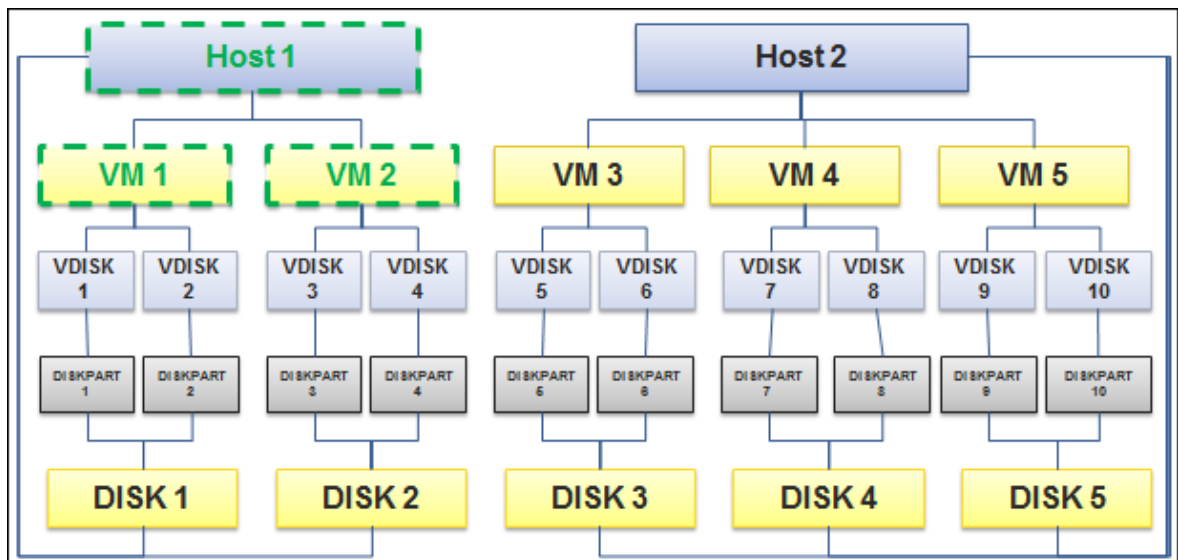
Example: `h[=name rx node AND osVersion rx 2003]`

### Scalar filter samples:

**NOTE:** In scalar filter samples, the dotted boxes show results of the query.

#### Sample 1: List Host1 and all the VMs that belong to Host1

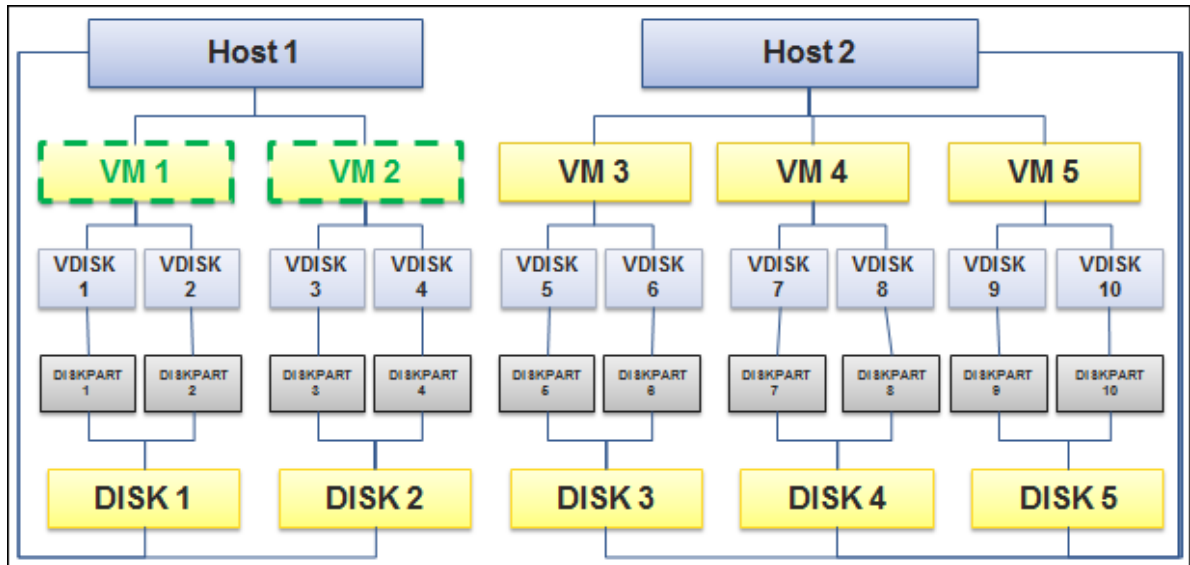
```
h[=name rx Host1]/vm
```



#### Sample 2: List all the VMs that belong to Host1

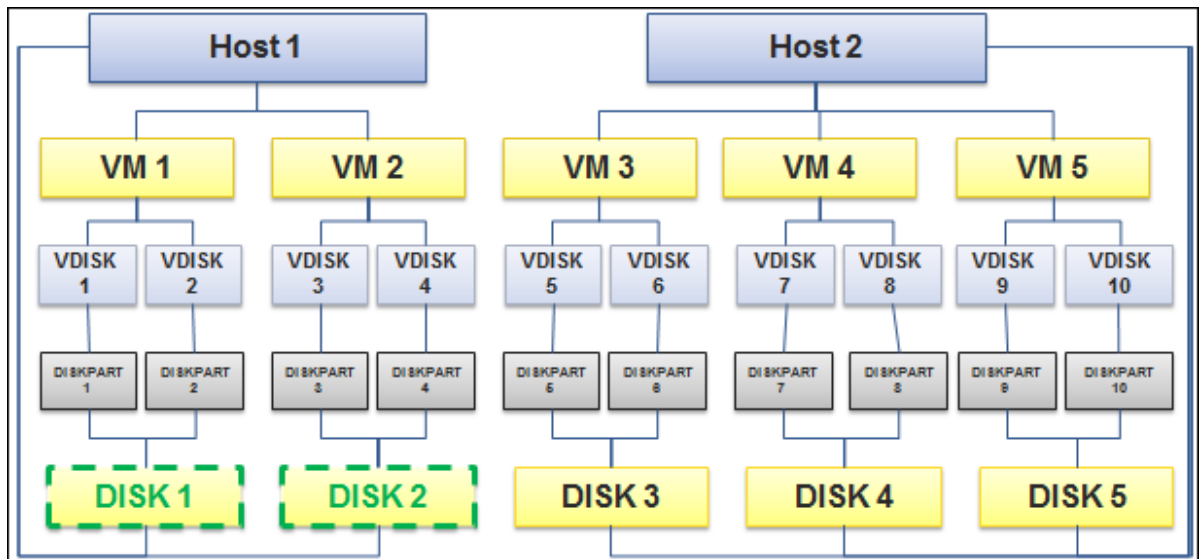
```
*h[=name rx Host1]/vm
```

**Note:** \* prefixed to resource type will cause the resource to be not displayed in the output



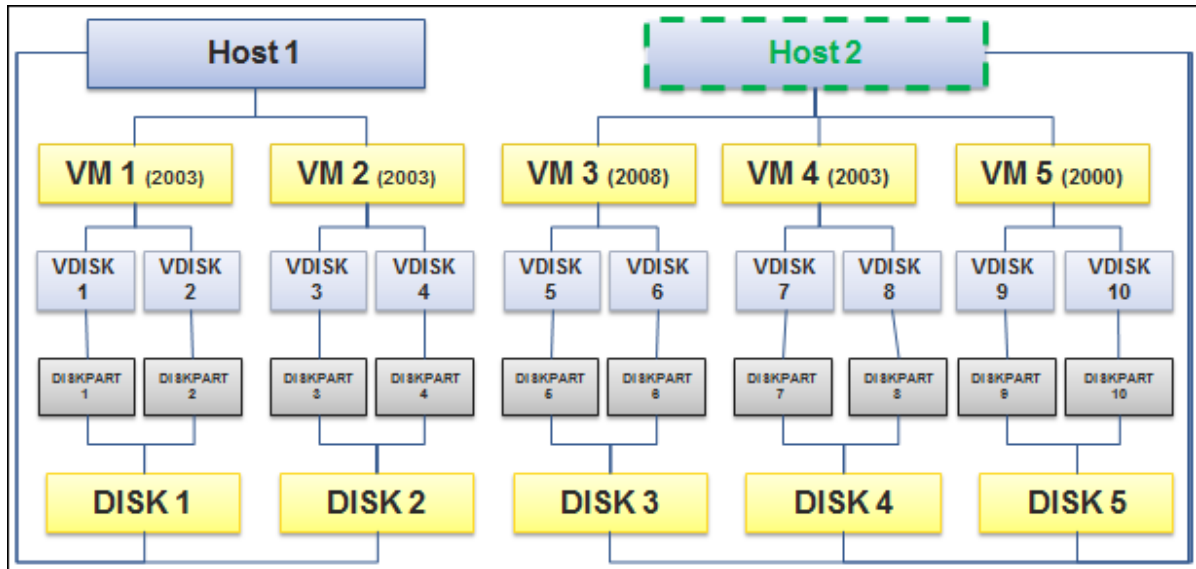
**Sample 3: List all Disks that belong to Host 1 and are connected to one or more VMs**

`*h[=name rx Host1]/*vm/*vdisk/*diskpart/disk`



**Sample 4: List all Hosts that have one or more VMs with OS Version 2000**

`h/&*vm[=osVersion rx 2000]`



### 3.1.4.2 Timeseries filters

A timeseries filter is used to filter resources based on their timeseries attribute values using modified regular expression syntax used for numerical matching. A basic timeseries filter consists of the timeseries attribute ID, look up type, and a regular expression, for example, `[@L_user rx b [U50-U100]{5,}]`.

The timeseries attribute ID can be a valid timeseries attribute applicable to that resource. The character 'b' specifies a basic lookup and character 'd' specifies delta lookup. Delta lookup calculates and returns the numerical difference of the current value in the database and its predecessor.

Any regular expression constructed as per JDK 1.7 specifications

(<http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html>) is supported with some modifications:

- All character and character class lookups are not supported. This is because timeseries does not consist of any characters. It only consists of numerical values.
- A character is substituted by a numerical value prefixed with character 'U'. This indicates to the query parser that a numerical value is specified, for example, U100 or U90.
- Similar to character range classes, numerical ranges can be specified, for example, [U90-U100].
- All quantifiers in a normal regular expression are supported, for example, [U90-U100]{5}.

#### Timeseries filter structure:

```
<resource_type>[@<attribute_id> rx b regex]
```

**Note:** Only numeric values are supported. Numeric values should be prefixed with 'U'.

### Examples:

- List all VMs with all cpuUsage  
vm[@cpuUsage rx b .\*]
- List VMs with cpuUsage = 60  
vm[@cpuUsage rx b [U60]]
- List VMs with cpuUsage value in the range of 70 and 100  
vm[@cpuUsage rx b [U70-U100]]

### Timeseries filter samples:

**NOTE:** In timeseries filter samples, the underlines show results of the query.

#### Sample 1: List all Hosts with cpuUsage between 90 and 100

```
h[@cpuUsage rx b [U90-U100]]
```

```
h#HostSig1,93,87,85,75,84,44,33,27,45,53,23,45,87,90,78,98,78,86,76,78,98,76,56,45,43,56,67,77,76,87
h#HostSig2,43,23,43,34,54,65,66,67,73,87,88,76,33,22,15,5,21,25,56,67,78,78,87,89,90,91,94,87,78,54
```

#### Sample 2: List all Hosts with cpuUsage between 90 and 100 with at least 3 consecutive occurrences

```
h[@cpuUsage rx b [U90-U100]{3,}]
```

```
h#HostSig1,93,87,85,75,84,44,33,27,45,53,23,45,87,90,78,98,78,86,76,78,98,76,56,45,43,56,67,77,76,87
h#HostSig2,43,23,43,34,54,65,66,67,73,87,88,76,33,22,15,5,21,25,56,67,78,78,87,89,90,91,94,87,78,54
```

#### Sample 3: List all Hosts with cpuUsage between 90 and 100 for 3 or more consecutive occurrences OR cpuUsagemhz between 70 and 100 for 5 or more consecutive occurrences

```
h[@cpuUsage rx b [U90-U100]{3,}][@cpuUsagemhz rx b [U70-U100]{5,}]
```

#### cpuUsage:

```
h#HostSig1,93,87,85,75,84,44,33,27,45,53,23,45,87,90,78,98,78,86,76,78,98,76,56,45,43,56,67,77,76,87
h#HostSig2,43,23,43,34,54,65,66,67,73,87,88,76,33,22,15,5,21,25,56,67,78,78,87,89,90,91,94,87,78,54
```

#### cpuUsagemhz:

```
h#HostSig1,83,77,75,65,64,54,53,57,55,53,43,35,47,60,68,78,78,86,76,88,98,86,76,65,63,56,57,47,46,37
h#HostSig2,93,83,83,74,74,55,56,67,53,57,38,36,33,32,45,45,41,45,46,47,58,58,57,69,60,61,64,57,58,34
```

In this case, the “OR” operator (default) is used between filters so it returns both Host1 and Host2.

**Sample 4: List all Hosts with cpuUsage between 90 and 100 for 3 or more consecutive occurrences AND cpuUsagemhz between 70 and 100 for 5 or more consecutive occurrences**

```
h[@cpuUsage rx b [U90-U100]{3,}]&[@cpuUsagemhz rx b [U70-U100]{5,}]
```

**cpuUsage:**

```
h#HostSig1,93,87,85,75,84,44,33,27,45,53,23,45,87,90,78,98,78,86,76,78,98,76,56,45,43,56,67,77,76,87
h#HostSig2,43,23,43,34,54,65,66,67,73,87,88,76,33,22,15,5,21,25,56,67,78,78,87,89,90,91,94,87,78,54
```

**cpuUsagemhz:**

```
h#HostSig1,83,77,75,65,64,54,53,57,55,53,43,35,47,60,68,78,78,86,76,88,98,86,76,65,63,56,57,47,46,37
h#HostSig2,93,83,83,74,74,55,56,67,53,57,38,36,33,32,45,45,41,45,46,47,58,58,57,69,60,61,64,57,58,34
```

In this case, the “AND” operator is used between filters so it returns Host2 only.

**Sample 5: List all Hosts with cpuUsage between 70 and 100 having at least one VM with cpuUsage between 60 and 100**

```
h[@cpuUsage rx b [U70-U100]]/&vm[@cpuUsage rx b [U60-U100]]
```

Only Host2 matches the query as VM5 belongs to Host 2. Host1 matches the cpuUsage filter but there is no qualified VM for it.

**cpuUsage - Host:**

```
h#HostSig1,93,87,85,75,84,44,33,27,45,53,23,45,87,90,78,98,78,86,76,78,98,76,56,45,43,56,67,77,76,87
h#HostSig2,43,23,43,34,54,65,66,67,73,87,88,76,33,22,15,5,21,25,56,67,78,78,87,89,90,91,94,87,78,54
```

**cpuUsage - VM:**

```
vm#VMSig1,23,27,25,25,24,24,23,27,25,23,23,25,27,20,28,28,28,26,26,28,28,26,26,25,23,26,27,27,26,27
vm#VMSig2,33,33,33,34,34,35,36,37,33,37,38,36,33,32,35,35,31,35,36,37,38,38,37,39,30,31,34,37,38,34
vm#VMSig3,43,47,45,45,44,44,43,47,45,43,43,45,47,40,48,48,48,46,46,48,48,46,46,45,43,46,47,47,46,47
vm#VMSig4,53,53,53,54,54,55,56,57,53,57,58,56,53,52,55,55,51,55,56,57,58,58,57,59,50,51,54,57,58,54
vm#VMSig5,63,67,65,65,64,64,63,67,65,63,63,65,67,60,68,68,68,66,66,68,68,66,66,65,63,66,67,67,66,67
```

### 3.1.4.2.1 Time windows

A timeseries filter is applied on a range of timeseries data as specified by the start time and end time of a query. The filter will match different subsequences within this range of timeseries data. These are called time windows. A group of such time windows can be given a name, for example:

```
lHost[@#tw#l_user rx b [U50-U100]{5,}].
```

The above example defines a time window named 'tw'. Just defining a time window does not actually alter the results. The results are altered when the time window is referred in another timeseries filter, for example:

```
lHost[@#tw#l_user rx b [U0-U50]{5,}][@^tw^l_iowait rx b [U50-U100]+]
```

Note the different syntax using '^' for referring to time windows. The above query matches all resources of type lHost where l\_user was between 0 and 50 for at least 5 consecutive data points, and in those time windows value of l\_iowait was between 50 and 100 for at least one data point. It is important to note that this matching is done on a resource-by-resource basis.

#### Matching across resource filters

Time window can be used across resource filters as well, for example:

```
lHost[@#tw#l_user rx b [U0-U50]{5,}]/lHostDisk[@^tw^l_svctm rx b [U50-U1000]+].
```

The above query matches all resources of type lHost where l\_user was between 0 and 50 for at least 5 consecutive data points. Value of l\_svctm attribute for all related lHostDisk was between 50 and 1000 for at least one data point within those time windows.

#### Time window filter samples:

**Sample 1: Return all Hosts with cpuUsage between 70 and 100 which have at least one VM with cpuUsage between 60 and 100**

```
h[@#tw#cpuUsage rx b [U70-U100]]/&vm[@^tw^cpuUsage rx b [U60-U100]]
```

In the following examples, a second filter (vm) uses the same time windows (tw) resulting from the first filter (h). It matches Host2 only as VM5 belongs to Host2. Host1 matches the cpuUsage data but there is no qualified VM for it.

#### cpuUsage – Host and VM

h#HostSig1	93	87	85	75	84	44	33	27	45	53	23	45	87	90	78	98	78	86	76	78	98	76	56	45	43	56	67	77	76	87
h#HostSig2	43	23	43	34	54	65	66	67	73	87	88	76	33	22	15	5	21	25	56	67	78	78	87	89	90	91	94	87	78	54
vm#VMSig1	23	27	25	25	24	24	23	27	25	23	23	25	27	20	28	28	28	26	26	28	28	26	26	25	23	26	27	27	26	27
vm#VMSig2	33	33	33	34	34	35	36	37	33	37	38	36	33	32	35	35	31	35	36	37	38	38	37	39	30	31	34	37	38	34
vm#VMSig3	43	47	45	45	44	44	43	47	45	43	43	45	47	40	48	48	48	46	46	48	48	46	46	45	43	46	47	47	46	47
vm#VMSig4	53	53	53	54	54	55	56	57	53	57	58	56	53	52	55	55	51	55	56	57	58	58	57	59	50	51	54	57	58	54
vm#VMSig5	63	67	65	65	64	64	63	67	65	63	63	65	67	60	68	68	68	66	66	68	68	66	66	65	63	66	67	67	66	67

### 3.1.4.2.2 Subsequence filtering

Time window based filtering does not further filter the subsequences matched. For example, consider [U90-U100]{5}, which resulted in 10 subsequences. Further filtering, as used in the previous section, either removes all subsequences if no match occurred in the lookup filter or will retain all the subsequences if a minimum of one match occurred.

To further filter on specific subsequences, additional syntax is available. Let's take the same query from previous section and add subsequence filtering to it.

```
lHost[@#tw#l_user rx b [U0-U50]{5,}](tw:ioWaitHigh = 1)[ioWaitHigh@^tw^l_iowait rx b [U50-U100]+]
```

First, notice that a 'filter tag' has been defined on the timeseries filter of attribute l\_iowait. Second, this filter tag has been referenced in a subsequence condition added to the timeseries filter of attribute l\_user.

A filter tag can be defined on any timeseries filter. The same tag can be used on multiple timeseries filters. It is referenced in a subsequence filter using the syntax <time window>:<filter tag>. Note, the filter tag reference is actually before the filter tag definition. This is the normal usage.

In the above example, for each matching lHost resource and its matching time windows of l\_user filter, results include only those subsequences on which the timeseries filter on l\_iowait also matches within the same start position and end position of the subsequence.

The subsequence expression value '1' in the filter means that at least one filter needs to match. By using the same filter tag multiple times, the matching can be made even stricter, for example:

```
lHost[@#tw#l_user rx b [U0-U50]{5,}](tw:ioWaitHigh = 2)[ioWaitHigh @^tw^l_iowait rx b [U50-U1000]+][ioWaitHigh @^tw^l_idle rx b [U70-U100]+]
```

The subsequence expression value '2' here means that both the filters (on l\_iowait and l\_idle) need to match for a subsequence of timeseries filters on l\_user to be matched.

#### Matching across resource filters

Subsequence filtering is allowed across resource filters. The syntax is the same as normal subsequence filtering. The difference is that the subsequence expression value takes a different meaning. Let's take the same multiple resource filter query from the time windows section and add subsequence filtering to it:

```
lHost[@#tw#l_user rx b [U0-U50]{5,}](tw:svctmHigh > 1)/lHostDisk[svctmHigh@^tw^l_svctm rx b [U50-U1000]+]
```

In the above example, for each matching lHost resource and its matching time windows of l\_user filter, results include only those subsequences in which at least two or more related lHostDisk resources filter match within the same start position and end position of the subsequence.

Therefore, unlike the previous case where the subsequence expression value referred to the number of filters, here it refers to the number of resources matched in other resource filters.



The subsequence expression value supports a few special values besides positive numerical values:

- 'a': All resources in the other resource filters must match. This could be used in the above example, to ensure that all IHostDisk related to that IHost match instead of two or more.
- -n: A negative value means all resources except 'n' must match.

By combining multiple time window definitions and lookups, and multiple filter tags and subsequence expressions, very complex and powerful filtering can be done on timeseries data.

## Subsequent filtering samples:

### Sample 1: List all Hosts with cpuUsage between 70 and 100 that have at least one VM with cpuUsage between 60 and 100.

```
h[@cpuUsage rx b [U70-U100]](t1 > 0)/vm[t1 @ cpuUsage rx b [U60-U100]]
```

In the following examples, the query matches to Host2 only as VM5 belongs to Host2. Host1 matches the cpuUsage data but there is no qualified VM for it.

#### cpuUsage - Host:

```
h#HostSig1,93,87,85,75,84,44,33,27,45,53,23,45,87,90,78,98,78,86,76,78,98,76,56,45,43,56,67,77,76,87
h#HostSig2,43,23,43,34,54,65,66,67,73,87,88,76,33,22,15,5,21,25,56,67,78,78,87,89,90,91,94,87,78,54
```

#### cpuUsage - VM

```
vm#VMSig1,23,27,25,25,24,24,23,27,25,23,23,25,27,20,28,28,28,26,26,28,28,26,26,25,23,26,27,27,26,27
vm#VMSig2,33,33,33,34,34,35,36,37,33,37,38,36,33,32,35,35,31,35,36,37,38,38,37,39,30,31,34,37,38,34
vm#VMSig3,43,47,45,45,44,44,43,47,45,43,43,45,47,40,48,48,48,46,46,48,48,46,46,45,43,46,47,47,46,47
vm#VMSig4,53,53,53,54,54,55,56,57,53,57,58,56,53,52,55,55,51,55,56,57,58,58,57,59,50,51,54,57,58,54
vm#VMSig5,63,67,65,65,64,64,63,67,65,63,63,65,67,60,68,68,68,66,66,68,68,66,66,65,63,66,67,67,66,67
```

### Sample 2: List all Hosts with cpuUsage between 70 and 100 that have all VMs with cpuUsage between 30 and 100

```
h[@cpuUsage rx b [U70-U100]](t1 = a)/vm[t1 @ cpuUsage rx b [U30-U100]]
```

In the following examples, the query matches Host 2 only as VM3, VM4, and VM5 belonging to Host 2 have qualified output. Host1 matches the cpuUsage data, but VM1 doesn't have a qualified output. Therefore, Host1 will not be qualified.

#### cpuUsage - Host:

```
h#HostSig1,93,87,85,75,84,44,33,27,45,53,23,45,87,90,78,98,78,86,76,78,98,76,56,45,43,56,67,77,76,87
h#HostSig2,43,23,43,34,54,65,66,67,73,87,88,76,33,22,15,5,21,25,56,67,78,78,87,89,90,91,94,87,78,54
```

#### cpuUsage - VM

```
vm#VMSig1,23,27,25,25,24,24,23,27,25,23,23,25,27,20,28,28,28,26,26,28,28,26,26,25,23,26,27,27,26,27
vm#VMSig2,33,33,33,34,34,35,36,37,33,37,38,36,33,32,35,35,31,35,36,37,38,38,37,39,30,31,34,37,38,34
vm#VMSig3,43,47,45,45,44,44,43,47,45,43,43,45,47,40,48,48,48,46,46,48,48,46,46,45,43,46,47,47,46,47
vm#VMSig4,53,53,53,54,54,55,56,57,53,57,58,56,53,52,55,55,51,55,56,57,58,58,57,59,50,51,54,57,58,54
vm#VMSig5,63,67,65,65,64,64,63,67,65,63,63,65,67,60,68,68,68,66,66,68,68,66,66,65,63,66,67,67,66,67
```

**Sample 3: List all Hosts with cpuUsage between 70 and 100 that have all but one VM with cpuUsage between 30 and 100.**

```
h[@cpuUsage rx b [U70-U100]](t1 = -1)/vm[t1 @ cpuUsage rx b [U30-U100]]
```

In the following examples, Host1 has 1 out of 2 VMs qualified = (all -1). Therefore, it will be qualified. Host2 has 3 out of 3 VMs qualified. Since all VMs qualified, Host2 will not be qualified. If 2 out of 3 VMs qualified, then Host2 would have been qualified.

**cpuUsage - Host:**

```
h#HostSig1,93,87,85,75,84,44,33,27,45,53,23,45,87,90,78,98,78,86,76,78,98,76,56,45,43,56,67,77,76,87
h#HostSig2,43,23,43,34,54,65,66,67,73,87,88,76,33,22,15,5,21,25,56,67,78,78,87,89,90,91,94,87,78,54
```

**cpuUsage – VM:**

```
vm#VMSig1,23,27,25,25,24,24,23,27,25,23,23,25,27,20,28,28,28,26,26,28,28,26,26,25,23,26,27,27,26,27
vm#VMSig2,33,33,33,34,34,35,36,37,33,37,38,36,33,32,35,35,31,35,36,37,38,38,37,39,30,31,34,37,38,34
vm#VMSig3,43,47,45,45,44,44,43,47,45,43,43,45,47,40,48,48,48,46,46,48,48,46,46,45,43,46,47,47,46,47
vm#VMSig4,53,53,53,54,54,55,56,57,53,57,58,56,53,52,55,55,51,55,56,57,58,58,57,59,50,51,54,57,58,54
vm#VMSig5,63,67,65,65,64,64,63,67,65,63,63,65,67,60,68,68,68,66,66,68,68,66,66,65,63,66,67,67,66,67
```

**Sample 4: List all Hosts with cpuUsage between 70 and 100 that have at least one VM with cpuUsage between 60 and 100 AND cpuUsagemhz between 70 and 100**

```
h[@cpuUsage rx b [U70-U100]](t1 > 0 & t2 > 0)/vm[t1 @ cpuUsage rx b [U60-U100]] [t2 @ cpuUsagemhz rx b [U70-U100]]
```

In the following examples, Host2 has at least one VM (VM5) with qualified output.

**cpuUsage - Host:**

```
h#HostSig1,93,87,85,75,84,44,33,27,45,53,23,45,87,90,78,98,78,86,76,78,98,76,56,45,43,56,67,77,76,87
h#HostSig2,43,23,43,34,54,65,66,67,73,87,88,76,33,22,15,5,21,25,56,67,78,78,87,89,90,91,94,87,78,54
```

**cpuUsage – VM:**

```
vm#VMSig1,23,27,25,25,24,24,23,27,25,23,23,25,27,20,28,28,28,26,26,28,28,26,26,25,23,26,27,27,26,27
vm#VMSig2,33,33,33,34,34,35,36,37,33,37,38,36,33,32,35,35,31,35,36,37,38,38,37,39,30,31,34,37,38,34
vm#VMSig3,43,47,45,45,44,44,43,47,45,43,43,45,47,40,48,48,48,46,46,48,48,46,46,45,43,46,47,47,46,47
vm#VMSig4,53,53,53,54,54,55,56,57,53,57,58,56,53,52,55,55,51,55,56,57,58,58,57,59,50,51,54,57,58,54
vm#VMSig5,63,67,65,65,64,64,63,67,65,63,63,65,67,60,68,68,68,66,66,68,68,66,66,65,63,66,67,67,66,67
```

**cpuUsagemhz –VM:**

```
vm#VMSig1,83,77,75,65,64,54,53,57,55,53,43,35,47,60,68,78,78,86,76,88,98,86,76,65,63,56,57,47,46,37
vm#VMSig2,93,83,83,74,74,55,56,67,53,57,38,36,33,32,45,45,41,45,46,47,58,58,57,69,60,61,64,57,58,34
vm#VMSig3,43,47,45,45,44,44,43,47,45,43,43,45,47,40,48,48,48,46,46,58,58,56,56,55,43,46,47,47,46,47
vm#VMSig4,63,63,63,64,64,65,66,67,63,67,68,66,63,62,65,65,61,65,66,67,68,68,67,69,60,61,64,67,68,64
vm#VMSig5,83,87,85,85,84,84,83,87,85,83,83,85,87,80,88,88,88,86,86,88,88,86,86,85,83,86,87,87,86,87
```

### 3.1.4.2.3 Scalar attribute as timeseries

The scalar attribute as timeseries feature allows you to query the scalar attribute as timeseries and shows the result as a series of data points.

If you want to see the configuration data at regular intervals, then you can query the scalar attributes as timeseries.

Syntax:

```
<R>[@<confAttr> rx b <regex>]
```

Where,

R is resource

confAttr is a scalar metric.

For example: `disk[@usedSpace rx b .*]`

This query returns the usedSpace configuration (scalar) data in a timeseries format at the interval of 1 minute.

**Note:** Currently, the result is shown in default interval of 1 minute.

# 4 Query language BNF

The part of query language syntax has been described in Backus-Naur Form with some extensions. The following notations are followed:

- ::= means definition of
- <abcd> means a non-terminal
- { ... } means zero or more occurrences
- [ ... ] means optional (zero or one occurrence)
- '...' means literal occurrence of the characters represented between quotes
- ... | ... means 'or'
- <ALLUPPERCASE>: A special non-terminal whose definition is provided outside of BNF
- White space is not allowed between non-terminals or literal occurrences. White space is allowed in syntax only where <SPACE> non-terminal has been explicitly mentioned.

## 4.1 Query statement

```
query_statement::=<resource_filter> { '/' [ '&' ] <resource_filter> }
resource::=[ '*' ]<RESOURCE_DEF_NAME><filters>
filters::= <attribute_filter>
           { '&'<attribute_filter> }
attribute_filter::=config_filter | time_series_filter
```

## 4.2 Scalar filter

```
config_filter::=[' ' '='<config_filter_def> {
<config_condition><config_filter_def> } ']'
config_filter_def::=<ATTRIBUTE_ID><SPACE>'rx'<SPACE><REGEX>
config_condition::='OR' | 'AND'
```

## 4.3 Timeseries filter

```
time_series_filter      ::= '[' [ '*' ] [ <filter_tags><SPACE> ] '@' [
<time_window> ]
<ATTRIBUTE_ID>'rx'<data_type><time_series_filter_expression>']'
[' '<subsequence_filter>']'
filter_tags::=<filter_tag> [ ', '<filter_tag> ]
filter_tag::=<STRING_LITERAL>
time_window::=<time_window_definition>|<time_window_reference>
time_window_definition::='#'<time_window_name>'#'
time_window_reference::='^'<time_window_name>'^'
time_window_name::=<STRING_LITERAL>
data_type::='b' | 'd'
```

### 4.3.1 Timeseries filter expression

```
time_series_filter_expression ::=    '.*' | { '.' }
'[' 'U' <INTEGER_LITERAL> '-' 'U' <INTEGER_LITERAL> ']' [
<occurrence_definition> ] { '.' }
|
<MODIFIED_REGEX>
occurrence_definition ::= '+' | '{' <NUMBER_LITERAL> ',' [ <NUMBER_LITERAL> ]
'}
```

### 4.4 Subsequence filter

```
subsequence_filter      ::=    '(' <subseq_expr> {
<SPACE> <subseq_condition_operator> <subseq_expr> } ')'
subseq_expr ::=
    <subseq_expr_variable> <SPACE> <subseq_expr_operator> <SPACE> <subseq
_expr_value>
subseq_expr_operator    ::=    '&' | '/'
subseq_expr_variable    ::=    <time_window_name> ':' <filter_tag>
subseq_expr_operator    ::=    '=' | '>' | '<' | '!'
subseq_expr_value ::=    <INTEGER_LITERAL> | 'a'
```

### 4.5 Non-terminals

REGEX: Any regular expression constructed as per JDK 1.7 specifications  
(<http://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html>)

MODIFIED\_REGEX: A normal regular expression but without string lookups and special syntax for numerical value lookups. Refer to the “Timeseries Filter” section for details.

INTEGER\_LITERAL: Any positive or negative number

NUMBER\_LITERAL: A positive number

STRING\_LITERAL: A string consisting of alphanumeric characters only

RESOURCE\_DEF\_NAME: Refer to the “Resource Definition” section for valid names

ATTRIBUTE\_DEF\_NAME: Refer to the “Resource Definition” section for the valid attribute name in the context of resource definition

SPACE: One or white spaces – blank and tab only

### 4.6 Limitations

To build a query, except \, all other special characters can be used if they make a meaningful regex.

# 5 Advanced data processing functions

Advanced data processing functions enable users to perform certain operations without any extra efforts. There are a set of in-built functions, which allow users to perform certain operations like aggregation, roll up, and deriving a new attribute.

Following are the supported advanced data processing functions:

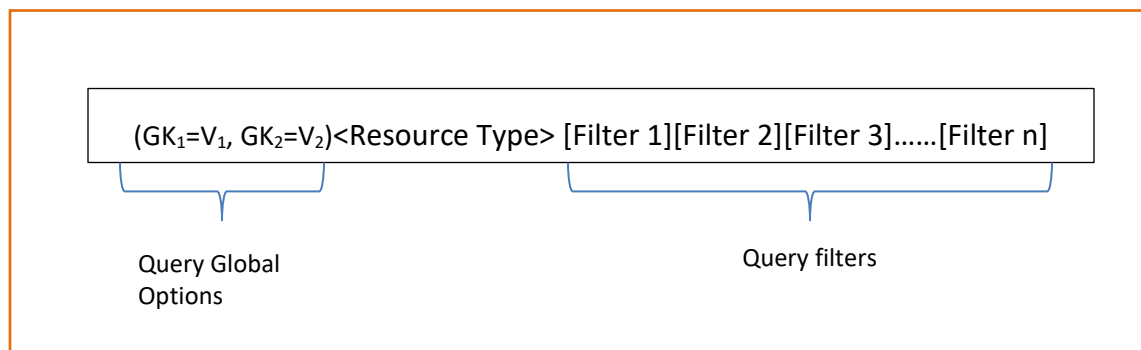
Function	Description
Interval rollup	Method to perform aggregation of timeseries data from lower to higher data interval.
Derived attributes	Method to create custom attribute using two or more attributes of a resource.
Resource rollup	Method to create custom attribute by aggregating data from related subresources to parent resource.
PreProc	Method to perform some specific task before actual processing of filter. For example, conditional processing of filter.

## 5.1 Query options

To use advanced data processing functions, the user needs to provide some additional options to MQL. These additional inputs are provided to MQL using Query Options. Query options can be categorized as Query Global Options and Query Filter Options.

### 5.1.1 Query global options

Query global options are the settings that influence the behavior of the entire query result. It is applicable to all the filters specified in the query and it must be provided in the beginning of query. The following gives the syntax of Query Global Options.



Where:

- (): Query Global Options are enclosed in parentheses at the beginning of query.
- (GK<sub>1</sub>=V<sub>1</sub>, GK<sub>2</sub>=V<sub>2</sub>): GK<sub>1</sub> and V<sub>1</sub> are a key-value pair. Multiple query global options can be separated by commas.

For example:

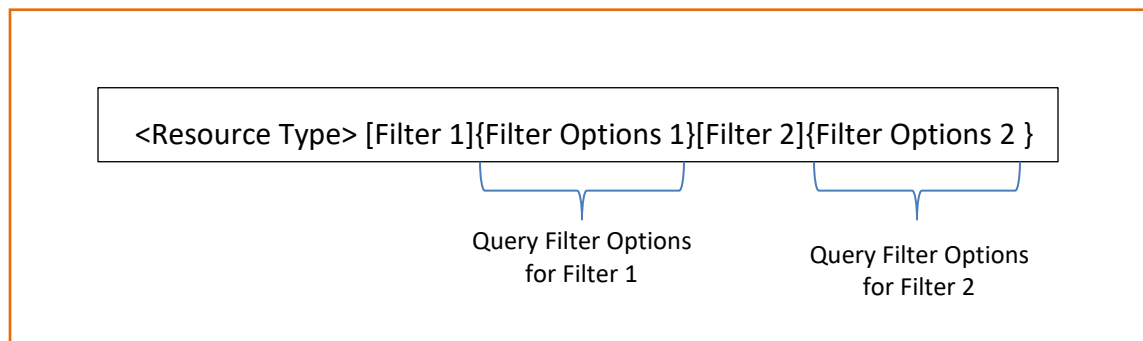
```
(useStrictAggr=true)vm[@cpuUsage rx b
.*]{intervalRollupOp=AVG,outputIntervalInSec=300}
```

Following are the supported Query Global Options:

Query Global Option	Description	Applicable for
useStrictAggr	This Boolean option is used to restrict partial data processing for generating output during interval rollup. The valid values are true and false.	Interval rollup
useRealTime	Helps define whether real-time data should be included in query processing when running a query from the Custom Reports feature in the server. The valid values are true and false.  If this option is not specified in the query, real-time data will be included in query processing by default.	All queries <b>Exception:</b> If a query includes multiple filters and one of those filters is the interval rollup filter, the real time will not be included in query processing.

## 5.1.2 Query filter options

Query filter options are the settings that influence the behavior of a single filter with which it is associated and must follow the associated filter. The syntax of the Query Filter Options is given below:



Where:

- [ ]: Query filter is enclosed in square brackets.
- { }: Query Filter Options are enclosed in curly braces. Each query filter can have Query Filter Options.

For example:

```
vm[@cpuUsage rx b .*]{intervalRollupOp=AVG,outputIntervalInSecs=300}
```

Following are the supported Query Filter Options:

Query Filter Options	Description	Applicable for
inputIntervalInSec	Optional parameter to specify which interval data will be used in calculation. It is useful in cases where data with multiple intervals is present in the Analyzer detail view database for the same time duration.	Interval Rollup/ Derived Attribute/ Resource Rollup
outputIntervalInSec	Parameter to specify the data interval for expected output.	Interval Rollup/ Derived Attribute/ Resource Rollup
intervalRollupOp	Operation to be used while performing interval rollup. Valid values are - SUM AVG MIN MAX	Interval Rollup
inputCounters	List of counter(s) that will be used in creation of output. Additionally, operation can also be specified in case of derived attributes. Valid values for derived attribute operations are - SUM AVG DIFF	Derived Attribute / Resource Rollup
resourceRollupOp	Operation to be used while performing resource rollup. Valid values are SUM AVG weightedAvg	Resource Rollup
outputUnit	Parameter to specify unit for newly created custom attribute.	Derived Attribute / Resource Rollup
preProc	Function to be performed before computing the filter.	PreProc

## 5.2 Interval Rollup

Interval Rollup is a technique to merge or aggregate data from a lower interval to a higher interval. Interval roll up is useful when the user is interested in low granularity of data interval.

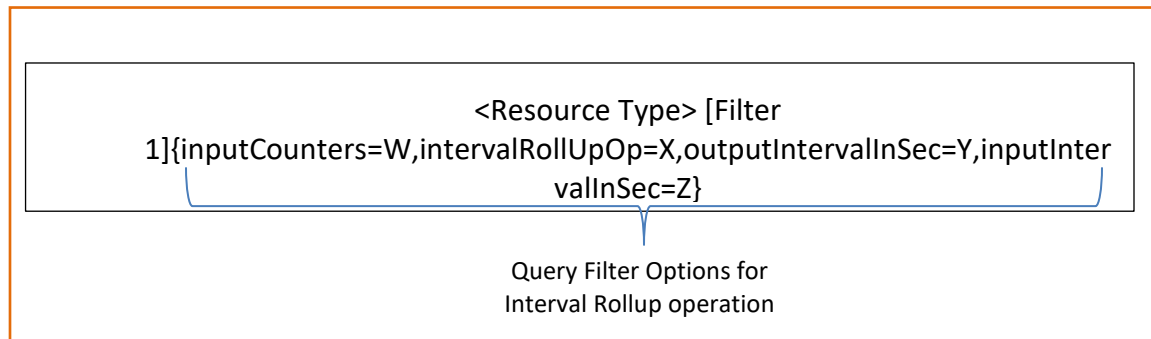


For example, the user has second-level timeseries data, but while analyzing data for a longer duration, the user might be interested in minute-level granularity. This can be achieved using interval rollup.

This function is applicable only for timeseries data.

**Note:** Input data interval must be less than or equal to 30 minutes for interval rollup operation.

## 5.2.1 Syntax



The following table describes query filter options for interval rollup:

Query Filter Options	Description	Valid Values
intervalRollupOp	Operation to be used for aggregation	SUM/AVG/MIN/MAX
outputIntervalInSec	Interval, in seconds, at which data needs to be rolled up	$0 < X \leq 60$ Where 60 is divisible by X
		$60 < X \leq 3600$ Where X is divisible by 60 & 3600 is divisible by X
		$3600 < X \leq 86400$ Where X is divisible by 3600 & 86400 is divisible by X
		$X > 86400$

		Where X is divisible by 86400
inputIntervalInSec	<p>This is optional.</p> <p>If multiple interval data exists, then this parameter can be used to specify the interval that should be used as input for the rollup operation.</p> <p>By default, data from all intervals will be flattened out and the result will be used as input for rollup.</p>	
inputCounters	<p>Attribute ID used for interval rollup operation.</p> <p>This is optional. By default, the counter specified in the associated timeseries filter will be used for interval rollup.</p>	W should be an existing predefine attribute.

## 5.2.2 Supported operations

The following table describes supported operations for interval Rollup:

Operation	Description
SUM	Addition of all non-data hole values will be reported as aggregated value.
AVG	Arithmetic mean of all non-data hole values will be reported as aggregated value. Data hole values will be ignored during calculation; only non-data holes will be added to get the numerator, and the denominator will also have a count of only non-data hole values.
MIN	Minimum of all non-data hole values will be reported as aggregated value.
MAX	Maximum of all non-data hole values will be reported as aggregated value.

## 5.2.3 Use case 1

The Analyzer detail view database has minute-level timeseries data for the cpuUsage attribute, but the user wants to analyze the same data on 5-minute or 10-minute level granularity.

By default, partial aggregation of data is allowed. The following table illustrates interval rollup behavior for the above-mentioned queries.

Query Start Time: 2:03:00													
Query		vm[@cpuUsage rx b .*]{intervalRollUpOp=AVG, outputIntervalInSecs=300}				vm[@cpuUsage rx b .*]{intervalRollUpOp=AVG, outputIntervalInSecs=600}							
Time	Data with interval 60 sec	outputIntervalInSec = 300				outputIntervalInSec = 600							
		SUM	AVG	MIN	MAX	SUM	AVG	MIN	MAX				
2:00:00		375.43	75.09	40.65	99.75	456.19	45.62	0.14	97.48				
2:01:00													
2:02:00													
2:03:00	56.50												
2:04:00	41.89												
2:05:00	95.07												
2:06:00	99.75												
2:07:00	40.65												
2:08:00	47.56												
2:09:00	92.40												
2:10:00	18.15	84.72	16.94	0.14	54.93	456.19	45.62	0.14	97.48				
2:11:00	0.14												
2:12:00	54.93												
2:13:00	3.84												
2:14:00	7.66												
2:15:00	97.48	371.47	74.29	63.98	97.48					456.19	45.62	0.14	97.48
2:16:00	71.53												
2:17:00	72.77												
2:18:00	65.72												
2:19:00	63.98												
2:20:00	74.46	263.48	65.87	54.56	74.83	263.48	65.87	54.56	74.83				
2:21:00	74.83												
2:22:00	59.63												
2:23:00	54.56												

Aggregated data will be reported at the aligned time intervals; for example, in the 10-minute interval it is aligned to 2:10

All the supported operations for interval rollup have been displayed in the above table for 5-minute and 10-minute rollup. Output data points are aligned to the nearest multiple of the output interval.

Data point at 2:00:00 is not computed using available partial data as the query start time is 2:03:00, which is after 2:00:00. Even though partial aggregation is allowed, moving back from the query time window is not implicit.

This point would have been calculated if the query start time was either at or before 2:00:00. This behavior is illustrated in the next table.

Query Start Time: 2:00:00									
Query		vm[@cpuUsage rx b .*]{intervalRollUpOp=AVG, outputIntervalInSecs=300}				vm[@cpuUsage rx b .*]{intervalRollUpOp=AVG, outputIntervalInSecs=600}			
Time	Data with interval 60 sec	outputIntervalInSec = 300				outputIntervalInSec = 600			
		SUM	AVG	MIN	MAX	SUM	AVG	MIN	MAX
2:00:00		98.39	49.20	41.89	56.50	473.82	67.69	40.65	99.75
2:01:00									
2:02:00									
2:03:00	56.50								
2:04:00	41.89								
2:05:00	95.07	375.43	75.09	40.65	99.75				
2:06:00	99.75								
2:07:00	40.65								
2:08:00	47.56								
2:09:00	92.40								
2:10:00	18.15	84.72	16.94	0.14	54.93	456.19	45.62	0.14	97.48
2:11:00	0.14								
2:12:00	54.93								
2:13:00	3.84								
2:14:00	7.66								
2:15:00	97.48	371.47	74.29	63.98	97.48				
2:16:00	71.53								
2:17:00	72.77								
2:18:00	65.72								
2:19:00	63.98								
2:20:00	74.46	263.48	65.87	54.56	74.83	263.48	65.87	54.56	74.83
2:21:00	74.83								
2:22:00	59.63								
2:23:00	54.56								

Since, query start time is at 2:00:00, the first data point for both aggregated intervals is computed using partial data.

The user can also force this initial aggregation by setting

***“query.allow.move.back.in.timewindow.for.aggr”*** to true in the query.properties file under /usr/local/megha/conf/sys/. This will enforce partial aggregation even if aligned start time is before query start time. The following table illustrates this behavior.

Query Start Time: 2:03:00 query.allow.move.back.in.timewindow.for.aggr: true									
Query		vm[@cpuUsage rx b .*]{intervalRollUpOp=AVG, outputIntervalInSecs=300}				vm[@cpuUsage rx b .*]{intervalRollUpOp=AVG, outputIntervalInSecs=600}			
Time	Data with interval 60 sec	Output Interval In Seconds = 300				outputIntervalInSec = 600			
		SUM	AVG	MIN	MAX	SUM	AVG	MIN	MAX
2:00:00		98.39	49.20	41.89	56.50	473.82	67.69	40.65	99.75
2:01:00									
2:02:00									
2:03:00	56.50								
2:04:00	41.89								
2:05:00	95.07	375.43	75.09	40.65	99.75				
2:06:00	99.75								
2:07:00	40.65								
2:08:00	47.56								
2:09:00	92.40								
2:10:00	18.15	84.72	16.94	0.14	54.93	456.19	45.62	0.14	97.48
2:11:00	0.14								
2:12:00	54.93								
2:13:00	3.84								
2:14:00	7.66								
2:15:00	97.48	371.47	74.29	63.98	97.48				
2:16:00	71.53								
2:17:00	72.77								
2:18:00	65.72								
2:19:00	63.98								
2:20:00	74.46	263.48	65.87	54.56	74.83	263.48	65.87	54.56	74.83
2:21:00	74.83								
2:22:00	59.63								
2:23:00	54.56								

Partial aggregation is a default behavior. However, this can be overridden by using Query Global Option. User may pass useStrictAggr=true option to restrict partial aggregation of data.

For example:

```
(useStrictAggr=true)vm[@cpuUsage rx b
.*]{intervalRollupOp=AVG,outputIntervalInSec=300}
```

Following table displays the behavior when useStrictAggr is set to true in Query Global Option.

Query Start Time: 2:00:00 useStrictAggr: true									
Query		vm[@cpuUsage rx b .*]{intervalRollupOp=AVG, outputIntervalInSecs=300}				vm[@cpuUsage rx b .*]{intervalRollupOp=AVG, outputIntervalInSecs=600}			
Time	Data with interval 60 sec	outputIntervalInSec= 300				outputIntervalInSec= 600			
		SUM	AVG	MIN	MAX	SUM	AVG	MIN	MAX
2:00:00		375.43	75.09	40.65	99.75	456.19	45.62	0.14	97.48
2:01:00									
2:02:00									
2:03:00	56.50								
2:04:00	41.89								
2:05:00	95.07	84.72	16.94	0.14	54.93	456.19	45.62	0.14	97.48
2:06:00	99.75								
2:07:00	40.65								
2:08:00	47.56								
2:09:00	92.40								
2:10:00	18.15	371.47	74.29	63.98	97.48				
2:11:00	0.14								
2:12:00	54.93								
2:13:00	3.84								
2:14:00	7.66								
2:15:00	97.48								
2:16:00	71.53								
2:17:00	72.77								
2:18:00	65.72								
2:19:00	63.98								
2:20:00	74.46								
2:21:00	74.83								
2:22:00	59.63								
2:23:00	54.56								

**Note:** If data of the multiple intervals is present in the Analyzer detail view database for a resource within the same time, then for interval rollup each interval's data will be used if inputIntervalInSec query filter option is not specified. Each interval data will be picked one by one; if there is any data hole after aggregation from data belonging to an interval, only then it will be filled by aggregated data using other intervals.

For example, the Analyzer detail view database have data for 1-minute and 5-minute intervals for the same duration, and the user has requested 10-minute aggregation using AVG operation - in this case all of these interval data will be used in aggregation.

1-minute data	5-minute data	Output after aggregation from 1-minute data	Final output
	97.07		79.22
	61.36		
99.34	65.53		
81.97			
91.27			
22.55			
32.54			
25.45	41.94		
15.48			
79.31			
34.22			
55.27			

First, 1-minute data is aggregated to get data at 10-minute interval data. Since there was no 1-minute data to be aggregated for the first 10 minutes, 5-minute interval data was used for aggregation in that period.

However, the sequence of intervals is not fixed. It depends on the order of intervals in stored data.

## 5.2.4 Use case 2

The Analyzer detail view database has minute-level timeseries data for *cpuUsage* attribute, but the user wants to analyze the same data on 5-minute or 10-minute level granularity by using a multiple interval rollup operation such as min, max, and avg.

The user can use multiple filter options for *cpuUsage* having different interval rollup operations. But in that case, the user will not be able to identify which series are computed by using which given operation. We can use the filter options property *inputCounters* to overcome this problem.

For example:

```
vm[@cpuUsage_avg rx b .*]{inputCounters=cpuUsage,  
intervalRollUpOp=AVG,outputIntervalInSecs=300}[@cpuUsage_min]{  
inputCounters=cpuUsage, intervalRollUpOp=MIN,outputIntervalInSecs=300}
```

Here, cpuUsage is used for interval rollup, and cpuUsage\_avg, and cpuUsage\_min to show the computed series.

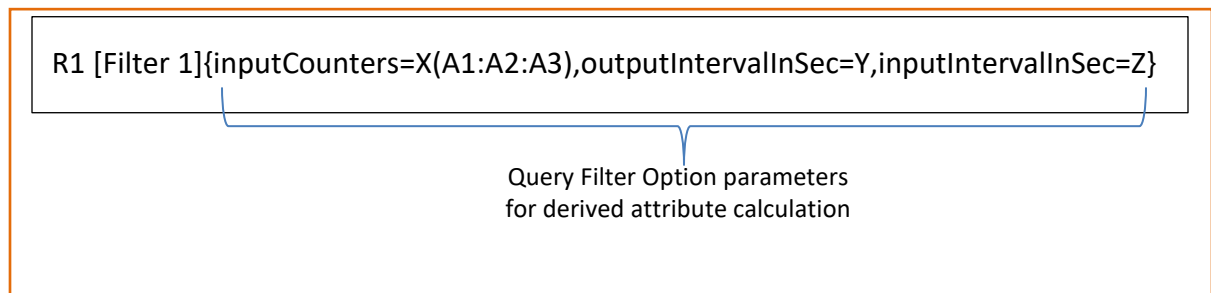
## 5.3 Derived attributes

New attributes can be computed on the fly from existing stored attributes within MQL.

MQL supports SUM, AVG, DIFF, MULT, DIV, and weightedAVG operations for the computation of derived attributes. These operations can be performed between timeseries attributes or scalar attributes. MULT and DIV operations can also be performed between:

- Timeseries attribute and constant numerical value
- Timeseries attribute and scalar attribute
- Scalar attribute and constant numerical value

### 5.3.1 Syntax



The following table describes query filter options for derived attributes:

Query Filter Options	Description	Valid Values	Remarks
inputCounters	Operation and list of attributes to be used for deriving new attribute	X can be SUM, AVG, DIFF, weightedAVG, MULT, DIV  A1, A2, A3 are attributes for Resource R1.	For DIFF, MULT, and DIV operations, only two attributes are permitted.



		<p>For the MULT and DIV operations, A2 can be a numeric constant value.</p> <p>MULT or DIV supports operations between timeseries and scalar attributes. If there is operation between timeseries and scalar attributes, A1 should be timeseries attribute and A2 should be scalar.</p>	
outputIntervalInSec	<p>This is optional.</p> <p>Interval, in seconds, at which output data is expected.</p> <p>If it is not specified, separate series for each available data interval or specified inputInterval will be computed and returned.</p>	Same as interval rollup	Only applicable for timeseries attributes
inputIntervalInSec	As explained above in interval rollup		Applicable only for timeseries attributes

## 5.3.2 Supported operations

The following table describes supported operations for derived attributes:

Operation	Description
SUM	Addition of non-data hole value for all inputCounter's attributes of a resource will be reported as a derived value.
AVG	Arithmetic mean of non-data hole value for all inputCounter's attributes of a resource will be reported as a derived value.
DIFF	DIFF operation supports two input counters. Subtraction of the second attribute's value from the first attribute value will be reported as a derived value.
weightedAvg	<a href="#">See use case 2</a>
MULT	The MULT operation supports two input counters. Multiplication of a non-data hole value for all inputCounter attributes of a resource are reported as a derived value.
DIV	The DIV operation supports two input counters. Division of a non-data hole value for all inputCounter attributes of a resource are reported as a derived value.

The above-mentioned operations can be performed on both time series and scalar attributes having numeric values. For example, Time series: IOPS, throughput, and so on. Scalar: Total capacity, Free capacity, and so on.

These operations on scalar attributes are supported only through Query API (REST API). Refer to, *Hitachi Ops Center Analyzer detail view REST API Reference Guide* for more information.

The Analyzer detail view's Report Builder feature does not support these operations on the scalar attributes.

## 5.3.3 Use case

### Use case 1

The Analyzer detail view database has `capacityGB` and `freeSpaceGB` attributes for each host, but the user wants to see the difference between these two as `usedcapacity`, which is not stored directly in the Analyzer detail view database.

```
ds[=usedcapacity rx .*]{inputCounters=DIFF(capacityGB:freeSpaceGB)}
```

The above query retrieves the scalar attributes `capacityGB` and `freeSpaceGB` for each host and reports the difference as `usedcapacity` on each.

For scalar attributes, allowed operations for derived attributes are explained in following table:

Resource	Scalar attribute 1	Scalar attribute 2	Operations				
			SUM	AVG	DIFF	MULT	DIV
R1	12.03	6.7	18.73	9.37	5.33	80.601	1.795522
R2	10.1	8.2	18.3	9.15	1.9	82.82	1.231707

For the DIFF operation, scalar attribute 2 is subtracted from scalar attribute 1. For instance, in the above query, `freeSpaceGB` will be subtracted from `capacityGB`.

For the DIV operation, scalar attribute 1 is divided by scalar attribute 2.

## Use case 2

Weighted average of attributes may be required where the general mean of data values is not sufficient, for example, while calculating a derived attribute from LDEV's attributes IOPS and `responseTime`. It would be meaningful to weight the response time on the basis of IOPS of LDEV at that time instead of taking the general average.

### Weighted average:

The weighted arithmetic mean is similar to an ordinary arithmetic mean (the most common type of average), except that instead of each data point contributing equally to the final average, some data points contribute more than others.

$$Avg_w = \frac{\sum_{i=1}^n I_n R_n}{\sum_{i=1}^n I_n}$$

Where,

***n*** = represents the number of data samples

***I*** = represents the weighted attribute

***R*** = represents the other attribute

Conditions for the derived weighted average operation:

Derived weighted average operation works for only two attributes of the same resource.

For example:

```
raidLdev[@weightedResponseTime rx b .*]{ inputCounters=weighted
AVG(IOPS:responseTime),inputIntervalInSec=60}
```

The above query computes the derived attribute "`weightedResponseTime`" by applying the weighted average operation on the IOPS and `responseTime` attributes.

The following two tables illustrate the weighted average operation on attribute I and R, where I is the weighted attribute:

The following table illustrates input data of attribute I and R for one resource:

Time	Res 1	
	I	R
2:00:00		
2:01:00	84.38	80.05
2:02:00	9.47	70.53
2:03:00	63.87	97.04
2:04:00	76.85	30.23

The following table shows intermediate results and weighted average:

Time	Res 1	$\Sigma I$	Weighted Average
	IR		
2:00:00			
2:01:00	6754.70	84.38	80.05
2:02:00	665.741	9.47	70.53
2:03:00	6197.9448	63.87	97.04
2:04:00	2323.1755	76.85	30.23

## Use case 2

The Analyzer detail view database has readIOPS and writeIOPS for each disk, but the requirement may be to analyze totalIOPS, which is not directly stored in the Analyzer detail view database.

```
disk[@totalIOPS rx b
.*]{inputCounters=SUM(readIOPS:writeIOPS),outputIntervalInSec=60}
```

The above query retrieves the timeseries data of attribute *readIOPS* and *writeIOPS* for each disk and reports the sum of these two as *totalIOPS* on each.

Similar to scalar derived attributes, timeseries attribute can also be derived using the available timeseries attribute on a resource. The following table displays different operations on two timeseries data for a resource to create a derived attribute. For SUM and AVG, more than two attributes can also be used.

Time	Counter 1 Data with interval 60 sec	Counter 2 Data with interval 60 sec	Operation				
			SUM	AVG	DIFF	MULT	DIV
2:00:00							
2:01:00							
2:02:00		95.07	95.07	95.07	-95.07		
2:03:00	56.50	99.75	156.25	78.13	-43.25	5635.88	0.57
2:04:00	41.89	40.65	82.54	41.27	1.24	1702.83	1.03

2:05:00	95.07	47.56	142.62	71.31	47.51	4521.53	2
2:06:00	99.75	92.40	192.15	96.08	7.36	9216.90	1.08
2:07:00	40.65	18.15	58.80	29.40	22.50	737.80	2.24
2:08:00	47.56	56.50	104.06	52.03	-8.94	2687.14	0.84
2:09:00	92.40	41.89	134.29	67.14	50.51	3870.64	2.21
2:10:00	18.15	95.07	113.22	56.61	-76.92	1725.52	0.19
2:11:00	0.14	99.75	99.89	49.94	-99.62	13.97	0
2:12:00	54.93	40.65	95.59	47.79	14.28	2232.90	1.35
2:13:00	3.84	47.56	51.40	25.70	-43.72	182.63	0.08
2:14:00	7.66	74.46	82.11	41.06	-66.80	570.36	0.1
2:15:00	97.48	74.83	172.31	86.15	22.65	7294.43	1.3
2:16:00	71.53	59.63	131.16	65.58	11.90	4265.33	1.2
2:17:00	72.77	54.56	127.34	63.67	18.21	3970.33	1.33
2:18:00	65.72	97.48	163.20	81.60	-31.76	6406.39	0.67
2:19:00	63.98	71.53	135.51	67.75	-7.55	4576.49	0.89
2:20:00	74.46	72.77	147.23	73.61	1.68	5418.45	1.02
2:21:00	74.83	65.72	140.55	70.27	9.11	4917.83	1.14
2:22:00	59.63	63.98	123.61	61.80	-4.34	3815.13	0.93
2:23:00	54.56	47.56	102.12	51.06	7.01	2594.87	1.15

DIFF and DIV operations on timeseries attributes work similarly to scalar attributes.

If outputIntervalInSecs is different from the data interval stored in the Analyzer detail view database and if no intervalRollupOp is specified, output will not be generated. If

OutputIntervalInSecs is different from the data interval, the user must specify intervalRollupOp.

This behavior is explained in following table:

Time	Counter 1 Data with interval 60 sec	Counter 2 Data with interval 60 sec	OutputIntervalInSec=300 (Interval Rollup Op Not Specified)			OutputIntervalInSec=300 Interval Rollup Op = SUM				
			SUM	AVG	DIF F	Counter 1 data rolled up	Counter 2 data rolled up	SUM	AVG	DIFF
2:00:00						-	-			
2:01:00										
2:02:00		95.07	-	-	-					
2:03:00	56.50	99.75	-	-	-					
2:04:00	41.89	40.65	-	-	-					
2:05:00	95.07	47.56	-	-	-	375.43	256.50	631.92	315.96	118.93
2:06:00	99.75	92.40	-	-	-					
2:07:00	40.65	18.15	-	-	-					
2:08:00	47.56	56.50	-	-	-					
2:09:00	92.40	41.89	-	-	-					
2:10:00	18.15	95.07	-	-	-	84.72	357.49	442.21	221.10	272.76
2:11:00	0.14	99.75	-	-	-					
2:12:00	54.93	40.65	-	-	-					
2:13:00	3.84	47.56	-	-	-					
2:14:00	7.66	74.46	-	-	-					
2:15:00	97.48	74.83	-	-	-	371.47	358.03	729.51	364.75	13.44
2:16:00	71.53	59.63	-	-	-					
2:17:00	72.77	54.56	-	-	-					

2:18:00	65.72	97.48	-	-	-					
2:19:00	63.98	71.53	-	-	-					
2:20:00	74.46	72.77	-	-	-					
2:21:00	74.83	65.72	-	-	-					
2:22:00	59.63	63.98	-	-	-					
2:23:00	54.56	47.56	-	-	-	263.48	250.02	513.51	256.75	13.46

### Use case 3

The Analyzer detail view database stores virtual volume capacity in GB (*vvolCapacityInGB*) for each logical device, but the user wants to analyze the total capacity in MB, which is not directly stored in the Analyzer detail view database.

```
raidLdev[@vvolCapacityInMB rx b
.*]{inputCounters=MULT(vvolCapacityInGB:
1000),outputIntervalInSec=60}
```

The above query retrieves the timeseries data of attribute *vvolCapacityInGB* for each logical device and reports the division *totalCapacityInGB* and 1000 as *totalCapacityInMB* on each of them.

The following table displays division and multiplication operations on timeseries data for a resource with numeric constant value to create a derived attribute.

Time	Counter 1 Data with interval 60 sec (Timeseries)	Operation	
		MULT by 1.5	DIV by 0.5
2:00:00			
2:01:00			
2:02:00			
2:03:00	56.50	84.75	113.00
2:04:00	41.89	62.84	83.78
2:05:00	95.07	142.61	190.14
2:06:00	99.75	149.63	199.50
2:07:00	40.65	60.98	81.30
2:08:00	47.56	71.34	95.12
2:09:00	92.40	138.60	184.80
2:10:00	18.15	27.23	36.30
2:11:00	0.14	0.21	0.28
2:12:00	54.93	82.40	109.86
2:13:00	3.84	5.76	7.68

2:14:00	7.66	11.49	15.32
2:15:00	97.48	146.22	194.96
2:16:00	71.53	107.30	143.06
2:17:00	72.77	109.16	145.54
2:18:00	65.72	98.58	131.44
2:19:00	63.98	95.97	127.96
2:20:00	74.46	111.69	148.92
2:21:00	74.83	112.25	149.66
2:22:00	59.63	89.45	119.26
2:23:00	54.56	81.84	109.12

#### Use case 4

The Analyzer detail view database stores speed in GB (*speedInGB*) for each port, but the user wants to see the port speed in MB, which is not stored directly in the Analyzer detail view database.

```
raidPort[=speedInMB rx .*]{inputCounters=MULT(speedInGB:1000)}
```

The above query retrieves scalar attributes' *speedInGB* for each port and reports the multiplication of *speedInGB* and 1000 as *speedInMB* on each of them.

For scalar attribute, allowed operations for derived attributes with constant numeric value are explained in the following table:

Resource	Scalar attribute 1	Operations with 1.5	
		MULT	DIV
R1	12.03	18.045	8.02
R2	10.1	15.15	6.733333

**Note:** Attribute ID specified for derived attributes may or may not be present in the Analyzer detail view database schema. However, it is recommended to use a predefined attribute ID for this operation.

## 5.4 Resource Rollup

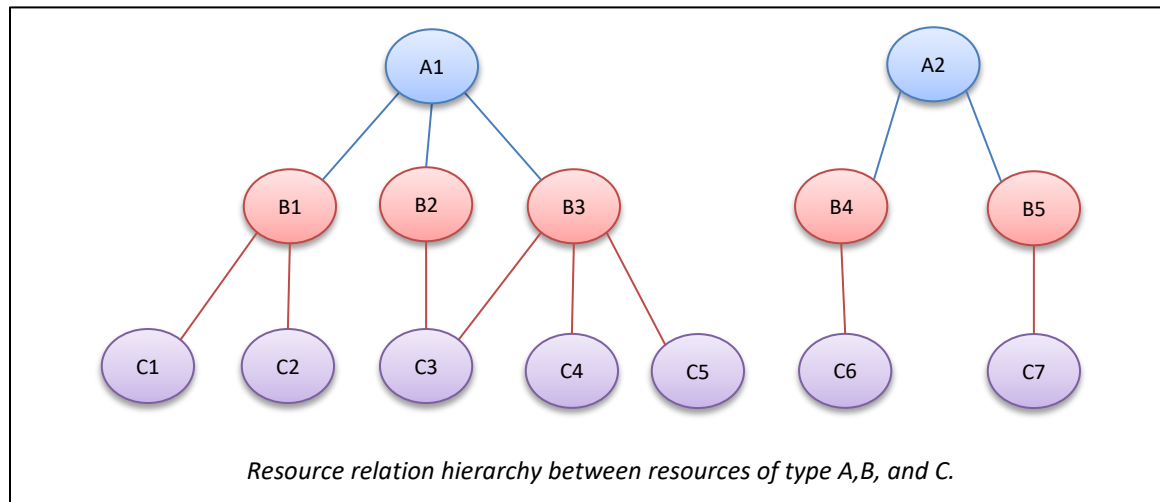
Analyzer detail view consolidates the attribute's value from all related resources at any level to the parent resource using specified attributes.

This can be useful in abstracting overwhelming details of individual resources by presenting data analysis on a higher level. Sometimes, the user needs to see the details at a higher level instead of its detailed view. For example, instead of looking into each LDEV counter, one might be interested to view it at the pool level. In this case, since the counter is actually present at the lower level, we should add up the values of all comprising resources to the upper level resource.



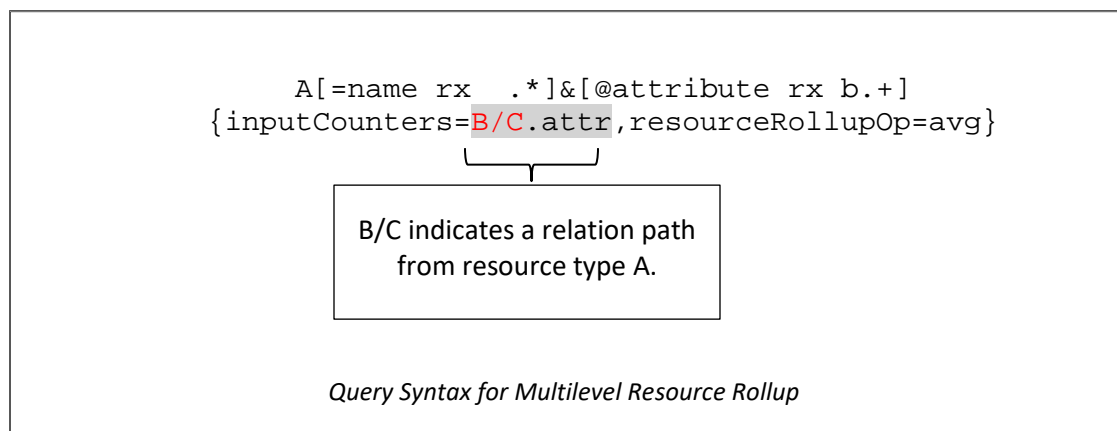
### Example:

Let us consider a scenario where resources from type A, B, and C are related as shown in the following diagram.



Using the Resource Rollup feature, the user can aggregate and view data from B1, B2, and B3 at A1 and B4, and B5 at A2 using query filter options. In addition, the user can also view data from C type resources at A1 and A2.

## 5.4.1 Syntax



The above query will roll up the timeseries attribute `attr` from all related resources of type C reachable from resources of type A via resources of B.

To explain the above statement let us consider relation hierarchy from the above diagram.

`attr` timeseries counters of C1, C2, C3, C4, and C5 will be rolled up to the A1 resource by averaging out data from all resources (C1, C2, C3, C4, and C5 ) at each timestamp. Even though

A1 has no direct relation with any of these resources of type C, they were reachable through intermediate resource B1, B2, and B3 of resource type B, as shown in the above diagram.

This feature allows the user to roll up data from any (directly or indirectly) related resources if a viable path exists between them.

As you can see, Resource C3 is reachable through two different paths:

A1 -> B2 -> C3, and

A1 -> B3 -> C3.

Measures will be taken by query processing to ensure that data is not considered twice (or more in case of more reachable paths).

*If any resource is reachable from more than one path for the same time window, data from that resource will be considered only **ONCE** for that time window to ensure data correctness.*

The following table describes query filter options for resource rollup:

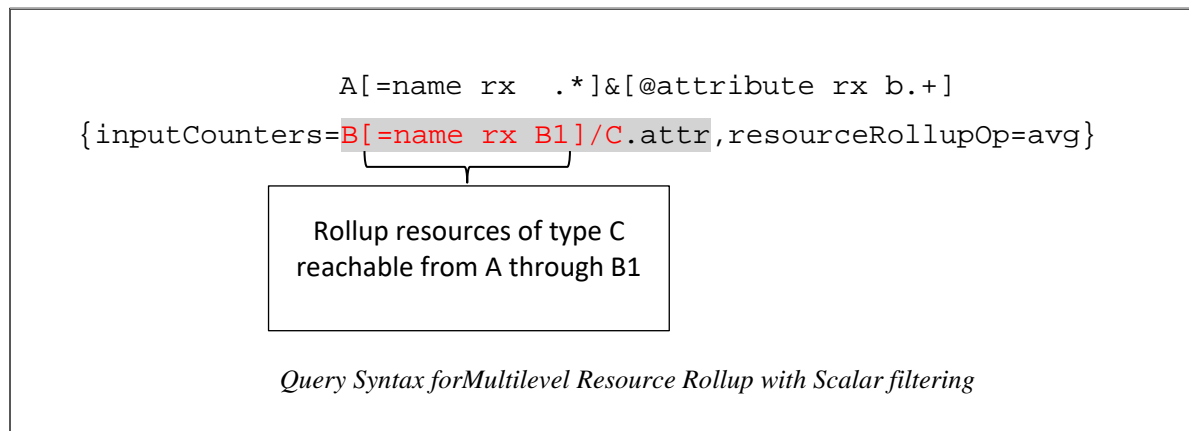
Query filter options	Description	Valid values	Remarks
resourceRollUpOp	Used for rolling up data from subresources	X can be SUM AVG MIN MAX weightedAvg	
inputCounters	List of attribute(s) and optional operations for rollup	B/C is a relation path from A.  'attr' is attribute of resource type C.	May be clubbed with Derived attribute if required – <Derived Operation>(B/C.attr1:attr2)
outputIntervalInSec	This is optional.  Same as interval rollup.  If it is not specified, separate series for each available data interval or specified inputInterval will be computed and returned.		Only applicable for timeseries attributes

Query filter options	Description	Valid values	Remarks
inputIntervalInSec	Same as interval rollup.		Only applicable for timeseries attributes

### 5.4.1.1 Resource filtering

There may be cases where the user does not want to view data being rolled up from every related subresource. One may want to apply certain filters to limit the resources which are used for resource rollup. To support this requirement, multilevel resource rollup allows Scalar and Relation filters to include only a subset of related resources in rollup computation.

#### Scalar Filtering



Scalar filters can be applied at any level in the relation path specified in the `inputCounters` parameter. This will restrict the reachable leaf-level resources from root.

For example, in the above scenario, due to scalar filtering at resource path B, the reachable C resources are only C1 and C2. Therefore, only timeseries data from these resources will be rolled up to A1. No data will be found at resource A2 as it is not related to resource B1.

#### Relation filtering

Under certain circumstances, the user may need to apply some filtering at the relationship level. For example:

*'Rollup resources of type C to A which are reachable through B and also are related to resource type D'.*

This need can be specified using Relation filtering.

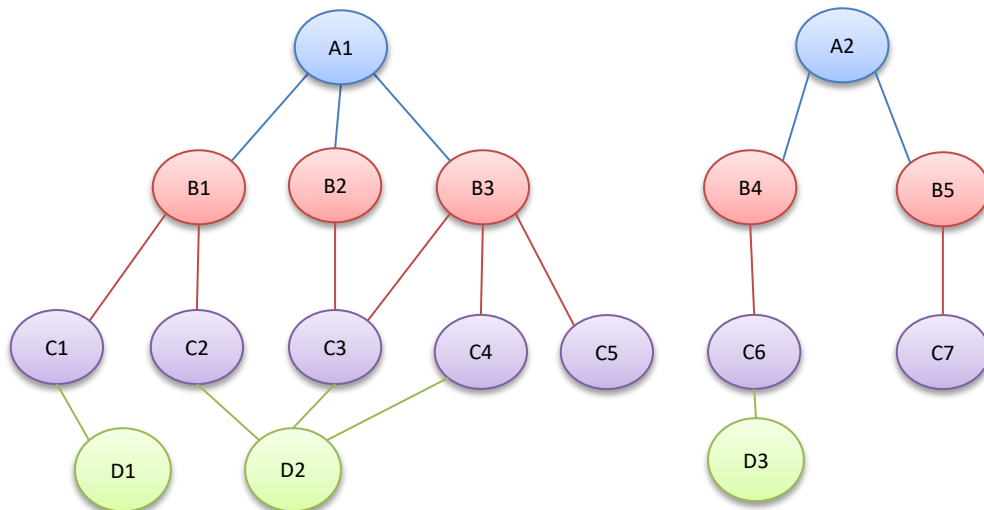
*Relation filters are newly added and are currently supported only inside multilevel resource rollup. They cannot be used outside Query filter options like other filters such as scalar and timeseries.*

```
A[=name rx  .*]&[@attribute rx b.+]
{inputCounters=B/C[/D.name rx .*].attr,resourceRollupOp=avg}
```

Only consider those resources  
of type C which are related to  
Resource of type D

*Query Syntax for Multilevel Resource Rollup with Relation filtering*

The above query may limit C resources from participating in resource rollup if they are not related to any resource of type D.



*Resource relation between resources of type A,B C, and D.*

Let us consider the above relation hierarchy to understand relation filtering,

As only C1, C2, C3, and C4 are reachable from A1 and also have relation with resource D1 and D2 of type D, only the data from these resources will be used in resource rollup. C5 will be filtered out even though it is reachable from A1 through B3 as it does not have relation with any resource of type D. Similarly, at A2, data from only C6 will be rolled up.

Multiple relation filtering or scalar filtering can be applied at any level. Use '&' for Logical AND of filters.

## 5.4.2 Supported operations

The following table describes supported operations for resource rollup:

Operation	Description
SUM	Addition of non-data hole values for each related subresource attribute.
AVG	Arithmetic mean of non-data hole values for each related subresource attribute.
MIN	Minimum of non-data hole values for each related subresource attribute.
MAX	Maximum of non-data hole values for each related subresource attribute.
weightedAvg	Explained later in use case 2.

The above operations can be performed on both timeseries and scalar attributes. Only scalar attributes having numerical values can be used.

## 5.4.3 Use case

### Use case: 1

The Analyzer detail view database has a pool with three associated LDEVs, each having corresponding utilization values. This can be rolled up at the pool level by averaging all LDEV utilization.

```
raidPool[@totalUtilization rx b .*]{resourceRollupOp=avg,  
inputCounters=raidLdev.utilization, inputIntervalInSec=60}
```

The above query rolls up LDEVs utilization to the pool level by averaging utilization.

The following table shows timeseries data of attribute 1 for three resources. These values can be rolled up to its parent resource using *resourceRollupOp*.

Time	Attribute 1			Parent resource	
	Resource 1	Resource 2	Resource 3	SUM	AVG
2:00:00					
2:01:00	56.50	99.75	18.15	174.40	58.13
2:02:00	41.89	40.65	56.50	139.04	46.35
2:03:00	95.07	47.56	41.89	184.52	61.51
2:04:00	99.75	92.40	95.07	287.22	95.74

*inputCounters* for resource rollup can also be derived from the subresource's attributes.

For example:

```
raidPool[@totalIOPS rx b .*] {resourceRollupOp=avg,  
inputCounters=SUM(raidLdev.readIOPS:writeIOPS), inputIntervalInSec=60}
```

In the above query, first *readIOPS* and *writeIOPS* will be added to get *totalIOPS* at each *raidLdev* level. Once *totalIOPS* is derived for each *raidLdev*, it will be rolled up to its parent pool using average operation.

The following table displays rollup of attributes derived from attribute1 and attribute2 present at the sublevel resource.

Time	attribute (C1)		Attribute(C 2)		Derived attribute = SUM(R.C1:C2)		Parent resource	
	Resource		Resource		Resource		SUM	AVG
	1	2	1	2	1	2		
2:00:00								
2:01:00	56.50	99.75	22.95	63.75	79.45	163.50	242.95	121.48
2:02:00	41.89	40.65	91.71	94.57	133.60	135.22	268.82	134.41
2:03:00	95.07	47.56	32.61	42.97	127.68	90.53	218.21	109.10
2:04:00	99.75	92.40	16.02	35.77	115.77	128.17	243.94	121.97

#### Use case 2:

Weighted average of attributes may be required where the general mean of data values is not sufficient, for example, while rolling up *responseTime* from LDEVs to pool. It would be meaningful to weight the response time on basis of IOPS of LDEV at that time instead of taking the general average.

#### Weighted average:

The weighted arithmetic mean is similar to an ordinary arithmetic mean (the most common type of average), except that instead of each of the data points contributing equally to the final average, some data points contribute more than others.

$$Avg_w = \frac{\sum_{i=1}^n I_n R_n}{\sum_{i=1}^n I_n}$$

Where:

*n* = number of data samples

*I* = represents the weighted attribute

*R* = represents the other attribute

Conditions for resource rollup weighted average operation:

- Weighted average rollup is a variant of resource rollup which requires two counters from a sublevel resource to contribute for each data point.
- Weighted rollup operation works for only two attributes.
- No need to specify derived operation in *inputCounters*.

For example:

```
raidPool[@weightedResponseTime rx b .*]{ resourceRollupOp=weightedAvg,
inputCounters=(raidLdev.IOPS:responseTime),inputIntervalInSec=60}
```

The above query rolls up LDEVs *responseTime* to pool level by cumulating each LDEV *responseTime* weighted by its IOPS.

The following two tables illustrate weighted average operation on attribute I and R where I is the weighted attribute.

The following table illustrates input data of attribute I and R for three resources.

Time	Res 1		Res 2		Res 3	
	I	R	I	R	I	R
2:00:00						
2:01:00	84.38	80.05	1.99	89.94	19.11	0.43
2:02:00	9.47	70.53	70.60	5.48	66.18	86.83
2:03:00	63.87	97.04	84.89	72.10	68.45	79.61
2:04:00	76.85	30.23	45.27	52.93	1.21	34.72

The following table shows intermediate results and weighted average.

Time	Res 1	Res 2	Res 3	$\Sigma IR$	$\Sigma I$	Weighted average
	IR	IR	IR			
2:00:00						
2:01:00	6754.70	179.19	8.31	6942.20	105.49	65.81
2:02:00	667.86	387.09	5745.73	6800.68	146.24	46.50
2:03:00	6197.30	6120.59	5449.69	17767.58	217.21	81.80
2:04:00	2322.89	2396.11	42.03	4761.03	123.32	38.61

**Note:**

- Resource rollup can be performed from any related resources.
- Analyzer detail view creates an attribute definition for rolled up attributes, if an attribute definition is not defined. However, it is recommended to use a predefined attribute ID for a rolled up attribute.
- In *weightedAvg*, attributes must be specified as *inputCounters* from the same subresource. In two attributes, the second attribute is weighed upon using the first attribute.

### Use case: 3

The Analyzer detail view database has a pool with 10 associated LDEVs, each having corresponding utilization values and only some have a parity group. So, if we want to consider only those LDEVs in rolled up which have a parity group, then the rolled up query will be:

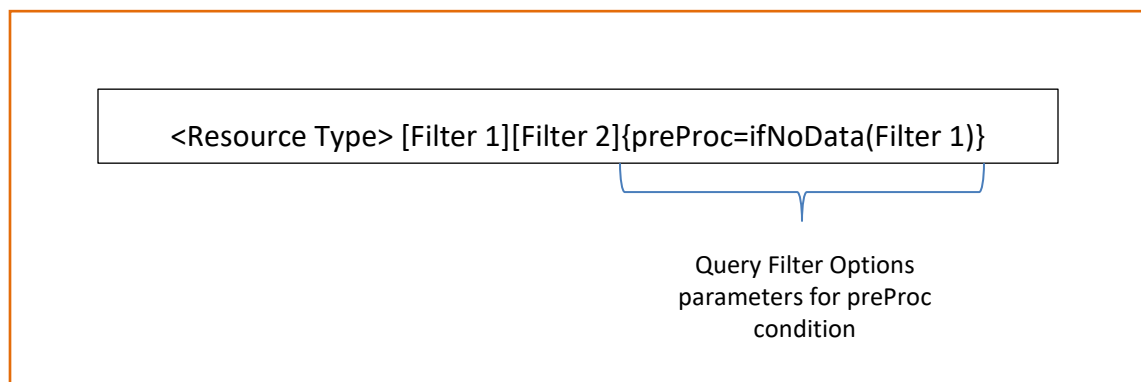
```
raidPool[@utilization rx b .+]{  
resourceRollupOp=avg,inputCounters=raidLdev[=parityGroup rx  
.+].utilization}
```

## 5.5 preProc

**PreProcessing** is a feature that allows performing a certain task before actually processing a filter. This allows making a decision such as whether or not to process the filter. It is useful for performing certain actions before evaluating filter output.

For example, there may be two attributes defined for a resource but either is required for analysis. In this case, output of one attribute filter is only computed if the other attribute value is not present in the Analyzer detail view database. To achieve this, the user can use the preProc condition as a query filter option.

### 5.5.1 Syntax



Where:

- **ifNoData**: Function that checks whether Filter1 result data is present or not. If Filter1 output data is present, then Filter 2 won't be processed.
- Presently only **ifNoData** method is supported for the preProc condition.

### 5.5.2 Use case

Data for the *totalIOPS* attribute might or might not be present for an LDEV resource depending on the collection mechanism. If it is not present, then compute by adding readIOPS and writeIOPS.



```
raidLdev[@totalIOPS rx b .*] [@derivedTotalIOPS rx b
.*]{preProc=ifNoData(totalIOPS),inputCounters=SUM(readIOPS:writeIOPS),outputIntervalInSec=60}
```

The above query will first compute *totalIOPS* for *raidLdev*, and it will calculate *derivedtotalIOPS* only if *totalIOPS* is not present in the Analyzer detail view database. *derivedtotalIOPS* will be derived by adding the *readIOPS* and *writeIOPS* attribute value.

# 6 External scripts

Functionality of MQL can be further extended by invoking scripts written in other programming languages. This is particularly useful when users want to leverage capabilities of other languages to satisfy their special needs. For example, R language has a built-in support for many statistical operations. Users can refer to R script in MQL, then the query engine would pass appropriate timeseries and scalar data to that R script, get it processed, and convert the output back in MQL format. In the following sections, we will look at supported programming languages, setup required, and syntax in detail.

However, the general mechanism of this functionality can be described as follows:

- User embeds R, Python, Groovy code/function/script in MQL to manipulate its scalar and timeseries output.
- A hook is provided in MQL that accepts a callback function in scripting language. This callback function manipulates original filter results and generates new output.
- This output is then passed back to the query engine and is made available as MQL output.

## 6.1 Supported scripting languages

Users can write custom processing functions in the following programming languages and use them from MQL:

- R
- Groovy
- Python

When MQL with external script is executed, the query engine connects/invokes appropriate language runtime, marshals data in the script's native data structures, executes the script, and then unmarshals script output back to MQL format.

To make this possible, scripts need to accept input and generate output in a certain format.

Details are discussed in the following sections.

### 6.1.1R

R is a programming language and software environment for statistical computing. The R language is widely used among statisticians and data miners for developing statistical software and data analysis.

R is separate software installed on the same or different machine. To use R we invoke R API remotely using TCP/IP server for R called RServe. RServe provides TCP/IP interface to R. RSession provides a nice API on top of RServe.

## 6.1.2 Groovy

Groovy is a powerful, optionally typed and dynamic language, with static-typing and static compilation capabilities, for the Java platform. It integrates smoothly with any Java program. It seamlessly integrates with all existing Java classes and libraries, and compiles straight to Java bytecode so no additional setup is required. Groovy has built-in support in the Analyzer detail view.

## 6.1.3 Python

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code. Python supports multiple programming paradigms, including object-oriented, imperative, and functional programming or procedural styles. It features a dynamic type system and automatic memory management, and has a large and comprehensive standard library.

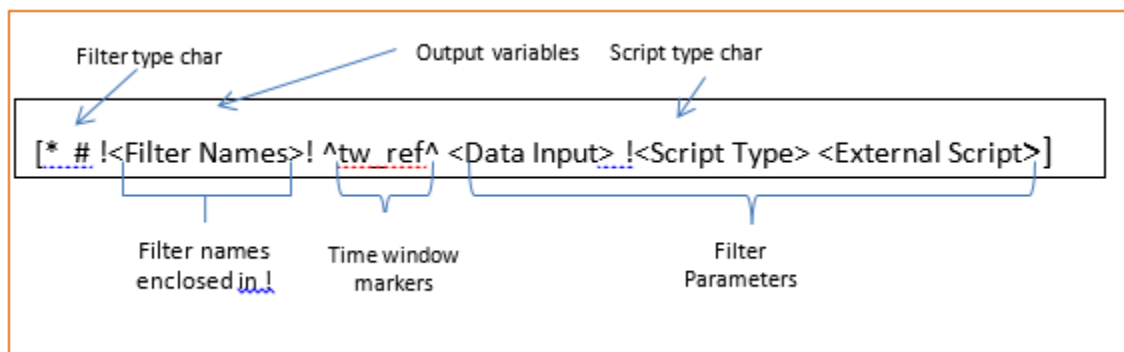
Marshalling and unmarshalling of objects between Java MQL and Python is supported using Thrift Framework.

### 6.1.3.1 Thrift framework

Thrift is an open-source interface definition language and binary communication protocol that is used to define and create services for numerous languages. It is used as a remote procedure call (RPC) framework and is developed for "scalable cross-language services development." It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between many languages.

## 6.2 Script filter

Script filter provides additional syntax to use external scripts from MQL.



Where:

- **Filter Names:** This is mandatory for script filter. Also note that it can have multiple comma-separated names. This will be used as output data variables which can be referenced in subsequent filters. More details will be provided in the following sections.
- **#:** Filter type character for script filter.

- **tw\_ref**: Time window reference which should be enclosed in **^**. The value should be the time window definition (tw\_def), which was added in the preceding filters. This is optional. The list of time windows will be made available to the external script. Time window reference is available to direct input variables of timeseries data but not for referenced data.
- **Data Input**: Data input variables that will be passed to the external script. Multiple inputs can be passed with comma separation. More details will be given in the following sections.
- **!**: Script type char is preceded by this special character “!”
- **Script Type**: Character to denote the external script type:
  - R – Language R
  - P – Python
  - G – Groovy
- **External Script**: External script code. It cannot include new line characters. If it uses any square brackets, such as “[ ]”, they should have the matching closing brackets.

### Script filter: data input

A comma-separated list of input data can be passed to the script. One or more data input is mandatory to the script. The following data can be passed to the script:

- Timeseries data of the resource. This variable must be preceded by **@**.
- Scalar data of the resource. This variable must be preceded by **=**.
- Data generated by the previous filters. This variable must be preceded by **:**.

### Example

```
h[!tFutureCpuUsage! # @cpuUsage, =cpuCount, :cpuUsageMhz !R script (P1, P2)
]
```

- **tFutureCpuUsage**: The output generated by the script will be assigned to this variable
- **@cpuUsage**: *cpuUsage* timeseries data of the host will be passed to the script
- **=cpuCount**: *cpuCount* scalar data of the host will be passed to the script
- **:cpuUsageMhz**: Referenced input generated by some other filter in the same query
- **!R**: “R” external script is used
- **script(P1, P2)**: R script to be executed with P1 and P2 parameters

### Script filter: data output

The data generated by the script will be assigned to data output variables. Script can generate multiple output data and in such cases, multiple output data variables should be specified as part of “Filter Names” in the beginning of the filter, which will be enclosed by the character **!**.

The first letter of the output variable will denote the data type of the output data. The following is the supported list of output data types:

- **Timeseries data**: Output data variable should be prefixed with the letter “t”
- **Vector data (list)**: Output data variable should be prefixed with the letter “v”
- **Scalar data**: Output data variable should be prefixed with the letter “s”

## Example

```
h[#!tFutureCpuUsage,sAverageCpuUsage! @cpuUsage, =cpuCount !R  
forecastCpuUsage]
```

- **tFutureCpuUsage**: Timeseries output data
- **sAverageCpuUsage**: Scalar output data

The Script filters can be further classified as:

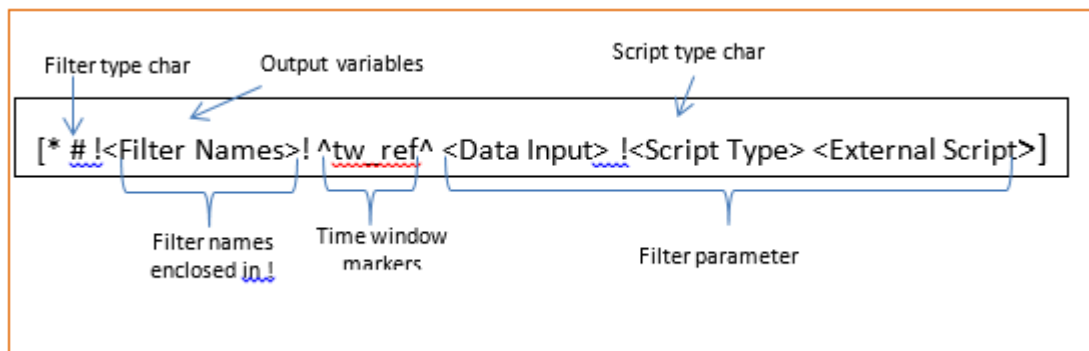
- **Regular Script Filter**: These are applicable to single-level resource data
- **Rollup Script Filter**: These are applicable when results from some resources need to be aggregated to the parent resource for a low granularity view.
- **Group Script Filter**: These are useful when the user needs to create a logical grouping of resources by a combination of some resources of the same type.

## 6.2.1 Regular external scripts

Regular script filters are useful in fulfilling requirements when data from resources needs to be modified to achieve results at the same level. It works on a single resource type and provides output at the same level.

### 6.2.1.1 Syntax

The syntax of the filter is as follows:



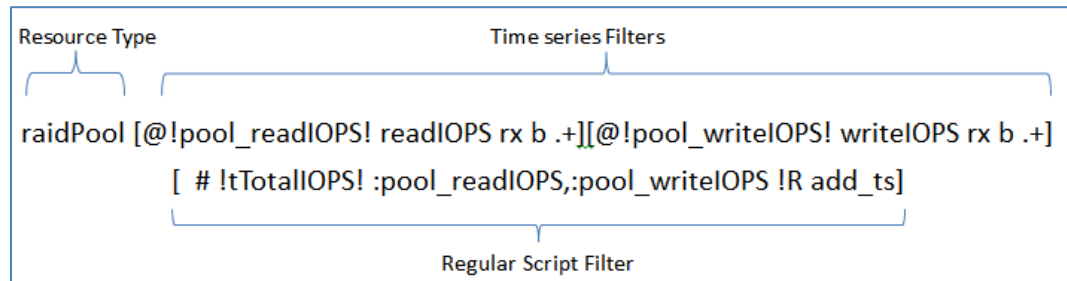
The input format and output format for data in scripts are predefined and are language specific. The following use case describes the input and output format for regular script in different scripting languages.

### 6.2.1.2 Use case

Calculate total IOPS for a resource by cumulating the readIOPS and writeIOPS attribute value for a resource.

## R

## Filter



## Input

Data is sent to the R environment (created using Rserve) in the form of lists. Each metric can be accessed in a list of its own. These names are available with the name given for input variables: *pool\_readIOPS* and *pool\_writeIOPS*.

The structure of the input list for *pool\_readIOPS* is as follows:

```
List of 4
$ attrId      : chr "readIOPS"
$ unit        : chr "Ops"
$ displayName: chr "Read IOPS"
$ perfList    :List of 3
..$ startTime: chr "2014-08-13 00:00:00"
..$ interval  : num 60000
..$ dataList  : int [1:50] 10 9 1 3 8 6 8 7 8 8 ...
```

Input data object can be of two types corresponding to both attribute types. Input data is classified as *timeseriesDataObject* for timeseries attributes and *scalarDataObject* for scalar attributes.

*timeseriesDataObject* is an RList and the following is the object structure:

Key	Data type	Description
unit	Char	Unit of measure
attrId	Char	Attribute ID
perfList	List	List of <i>timeseries</i>
displayName	Char	Attribute name

*timeseries* is an RList and the following is the object structure:

Key	Data type	Description
dataList	Vector	Vector of double values
startTime	Char	Data start time in format “yyyy-MM-dd HH:mm:ss”

Key	Data type	Description
interval	Integer	Data interval, in milliseconds

*scalarDataObject* is an RList and the following is the object structure:

Key	Data type	Description
unit	Char	Unit of measure
attrId	Char	Attribute ID
data	Char	Scalar value
dataType	Char	Double/String
displayName	Char	Attribute name

### Processing

The following is a sample R script to add two timeseries data inputs. First, it gets *perfList*, which itself is a list containing the actual timeseries, out of data inputs and then passes both timeseries lists to a function named *function\_to\_add*, which returns a new *perfList*. Users can write their own logic to add two timeseries data inputs in function *function\_to\_add*.

```
# get list of time-series map for pool_readIOPS
readIOPS_perf <- pool_readIOPS$perfList

# get list of time-series map for pool_writeIOPS
writeIOPS_perf <- pool_writeIOPS$perfList

#pass both of them to a function to perform required operations
totalIOPS <- function_to_add(readIOPS_perf, writeIOPS_perf)

# function to add time-series
function_to_add <- function(series1, series2) {
  # code to add the series
}
```

### Output

In R-based Regular script, output needs to be created as a list object for every output with an assigned variable name for each of them (outputs) as specified in Query. Structure of the output list is similar to the input list.

```

# prepare time-series
start <- "2014-08-13 00:00:00" #start time of resultant time series
interval <- 60000 #interval of timeseries in milliseconds
newdata <- function_to_add(readIOPS_perf, writeIOPS_perf) #list of double values
newseries<-list(startTime=start,interval=interval ,dataList=newdata)

# list of series
dataVector[[length(dataVector) + 1]] <- newseries

# create output result
tTotalIOPS[['attrId']] <- 'totalIOPS' #unique id for result time series
tTotalIOPS[['unit']] <- 'Ops' #unit for new attribute
tTotalIOPS[['displayName']] <- 'Total IOPS' #display name
tTotalIOPS[['dataType']] <- 'Timeseries' #DataType
tTotalIOPS[['data']] <- dataVector #list of timeseries

```

Structure of output RList is as follows:

```

List of 5
 $ attrId      : chr "totalIOPS"
 $ unit        : chr "Ops"
 $ displayName : chr "Total IOPS"
 $ dataType    : chr "Timeseries"
 $ data        : List of 1
 ..$ :List of 3
 .. ..$ startTime: chr "2014-04-13 00:00:00"
 .. ..$ interval : num 60000
 .. ..$ dataList  : num [1:80] 33 34 70 54 76 82 29 55 24 14 ...

```

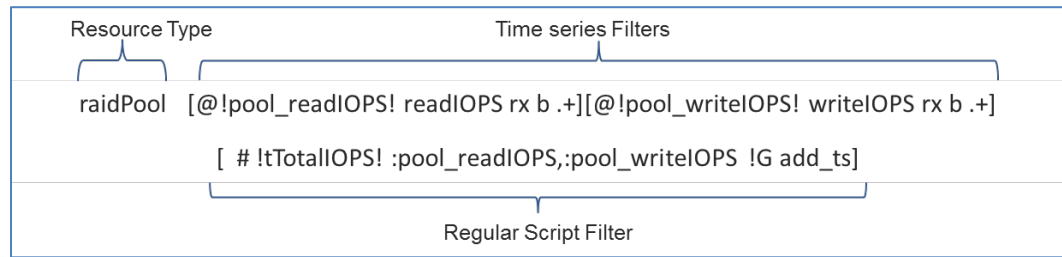
The following table illustrates object structure of output data, and value type of data varies with *dataType* of the output attribute:

Key	Data type	Description
unit	Char	Unit of measure
displayName	Char	Attribute name
attrId	Char	Attribute ID
dataType	Char	Data type and possible values are Timeseries/Double/String
data	List	If output is Timeseries, list of <i>timeseries</i>
	Vector	If output is vector, vector of Double/Char
	Double/Char	If output is scalar, Double / Char value



## Groovy

### Filter



### Input

Query engine makes data inputs available inside `add_ts.groovy` as groovy data structures. Data inputs are available with a name given for input variables: `pool_readIOPS` and `pool_writeIOPS`.

The structure of input list for `pool_readIOPS` is as follows:

```
pool_readIOPS :: [ unit:Ops,
                   attrId:readIOPS,
                   perfList:[
                       [ dataList:[0.0, 2.0, 2.0, ..., 4.0, 7.0, 0.0],
                         startTime:2014-08-13 00:00:00,
                         interval:60000
                       ],
                   ],
                   displayName:Read IOPS
                 ]
```

Each input data is a map and its object structure depends on the attribute type. Input data object can be of two types corresponding to both attribute types. Input data is classified as *timeseriesDataObject* for timeseries attributes and *scalarDataObject* for scalar attributes.

*timeseriesDataObject* is a map and the following is the object structure:

Key	Data type	Description
unit	String	Unit of measure
attrId	String	Attribute ID
perfList	List	List of <i>timeseries</i>
displayName	String	Attribute name

*timeseries* is a map and the following is the object structure:

Key	Data type	Description
dataList	List	List of double values
startTime	String	Data start time in format "yyyy-MM-dd HH:mm:ss"

Key	Data type	Description
interval	Integer	Data interval, in milliseconds

*scalarDataObject* is a map and the following is the object structure:

Key	Data type	Description
unit	String	Unit of measure
attrId	String	Attribute ID
dataType	String	Data value type and possible values are Double/String
data	Object	Data value
displayName	String	Attribute name

### Processing

The following is a sample groovy script to add two timeseries data inputs. First, it gets *perfList*, which itself is a list containing the actual timeseries, out of data inputs and then passes both timeseries lists to a function named as *function\_to\_add* which returns a new *perfList*. Users can write their own logic to add two timeseries data inputs in function *function\_to\_add*.

```
// get list of time-series maps from pool_readIOPS
readIOPS_perf_list = pool_readIOPS.get("perfList");

// get list of time-series maps from pool_writeIOPS
writeIOPS_perf_list = pool_writeIOPS.get("perfList");

//pass both of them to a function to perform required Operation
total_IO = function_to_add(readIOPS_perf_list, writeIOPS_perf_list)

//function to add time series
def function_to_add(def _readIOPS_perf_list,def _writeIOPS_perf_list) {
    // code to add the series.
}
```

### Output

For groovy scripts, an *outputMap* is created to hold all the output being generated by script, where key is the name of the output variable specified in the query, and value is the data structure holding actual data output. This output data object has a similar structure as input data.

The following groovy code snippet generates data output for 'totalIOPS' and adds 'totalIOPS' in *outputMap*:

```

outputMap = new HashMap<String,Object>();

def perfList = new ArrayList<Map<String,Object>>();

//prepare time-series
def dataMap = new HashMap<String,Object>();
dataMap.put("startTime", "2014-08-13 00:00:00"); //start time of resultant time series
dataMap.put("interval", 60000); // interval of time series in milliseconds
dataMap.put("dataList",dataList); // list of double values

perfList.add(dataMap); // list of time-series

// create output result map
totalIODataMap = new HashMap<String,Object>();
totalIODataMap.put("unit", "Ops"); // unit for new attribute
totalIODataMap.put("displayName", "Total IO"); // display name for resultant time-series
totalIODataMap.put("attrId", "totalIOPS"); // unique Id for time series
totalIODataMap.put("dataType", "Timeseries"); // valid values are Timeseries | Double | String
totalIODataMap.put("data",perfList); // list of time-series

// the output result map should be stored in outputMap against a key,
// where key name is equal to the outputName specified in query
// the first't' in key denotes that output is timeseries,
//it is mandatory to specify type in first letter of keyname (similarly 's' for scalar and 'v' for vector outputs)
outputMap.put("ttotalIO", totalIODataMap);

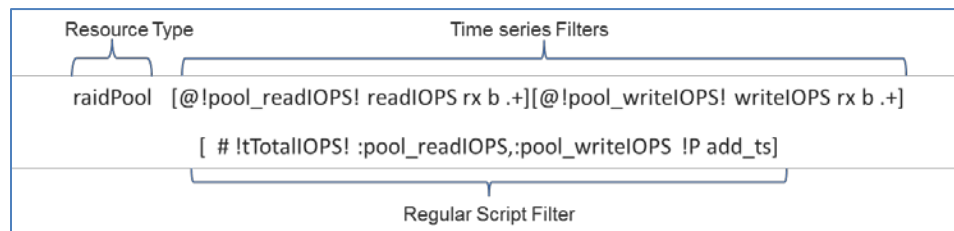
```

Object structure of output data is given below, and the value type of data varies with *dataType* of the output attribute:

Key	Data type	Description
unit	String	Unit of measure
displayName	String	Attribute name
attrId	String	Attribute ID
dataType	String	Data type and possible values are Timeseries/Double/String
data	List	If output is Timeseries, list of <i>timeseries</i>
	List	If output is vector, list of Double/String
	Object	If output is scalar, data value

## Python

### Filter



### Input

For Python scripts, inputs are thrift-implemented objects, and these thrift stub objects can be used to retrieve data in Python, process them, and send back the script output. The user needs to fetch data from these thrift objects and generate script outputs as thrift objects.

Inputs for regular scripts are objects of *ThriftPerfData*. Following is the object structure of *ThriftPerfData*:

Key	Type	Description
unit	String	Unit for timeseries data
displayName	String	Display name of timeseries data
perfList	List	List of <i>ThriftEPerfData</i> objects.

*ThriftEPerfData* contains data series, interval, and start time. Following is object structure of *ThriftEPerfData*:

Attributes	Type	Description
dataList	List	List of double values
startTime	String	Start time of data in format “yyyy-MM-dd HH:mm:ss”
interval	Long	Data interval, in milliseconds

Data inputs to script are available with the input variable name specified in the query: pool\_readIOPS and pool\_writeIOPS

### Processing

Starting point for script execution is *execute* method, and input data is available through its argument.

The following is a sample Python script to add two timeseries data inputs. First, it gets perfList, which itself is a list containing the actual timeseries, out of data inputs and then passes both timeseries lists to a function named as *function\_to\_add*, which returns a new perfList. Users can write their own logic to add two timeseries data inputs in function *function\_to\_add*.

```

import Thrift mql types
sys.path.append('/usr/local/megha/ExternalScript/thrift/gen-py')

from mql.ttypes import *

#implement execute method
def execute(**kargs) :
    # fetch readIOPS object
    readIOPS = kargs['pool_readIOPS']
    # get perf list
    readIOPS_perf = readIOPS.perfList
    # fetch writeIOPS perfList
    writeIOPS_perf = kargs['pool_writeIOPS'].perfList

    # implement Business Logic to calculate totalIOPS
    totalIOPS = function_to_add(readIOPS_perf, writeIOPS_perf)

    # add totalIOPS to out and return
    out['tTotalIOPS'] = totalIOPS
    return out

def function_to_add(**readIOPS_perf, **writeIOPS_perf) :
    # code to add series

```

## Output

Execute method returns a dictionary having output data corresponding to output variable name, *tTotalIOPS*. The following tables describe output data type based on data type mentioned in the data output variable:

If data type is	Output object type
Timeseries	<i>ThriftPerfData</i>
Scalar	<i>ThriftDoubleScalar/ThriftStringScalar</i>
vector	<i>ThriftDoubleVector/ThriftStringVector</i>

***ThriftPerfData*** contains the timeseries data of an attribute. It is explained in the input section.

***ThriftDoubleScalar*** contains the attribute unit, attribute's display name, and data. The following is the object structure of *ThriftDoubleScalar*.

Key	Type	Description
unit	String	Unit for scalar data
displayName	String	Display name of scalar data

Key	Type	Description
data	Double	Double scalar value

***ThriftStringScalar*** contains the attribute unit, attribute's display name, and data. Object structure of *ThriftStringScalar* is similar to *ThriftDoubleScalar* where the type of data is different. The following is its object structure:

Key	Type	Description
unit	String	Unit for scalar data
displayName	String	Display name of scalar data
data	String	String scalar value

***ThriftDoubleVector*** contains the attribute unit, attribute's display name, and data. The following is the object structure:

Key	Type	Description
unit	String	Unit for scalar data
displayName	String	Display name of scalar data
data	List	Double data vector

***ThriftStringVector*** contains the attribute unit, attribute's display name, and data. Object structure of *ThriftStringVector* is similar to *ThriftDoubleVector* where the type of data is different. The following is its object structure:

Key	Type	Description
unit	String	Unit for scalar data
displayName	String	Display name of scalar data
data	List	String data vector

The following Python script snippet prepares output data and adds output data in the out dictionary.

```

# list to hold ThriftEPerf Objects
epdList = [];
# where function_to_add returns list of double values
dataList = function_to_add(readIOPS_perf, writeIOPS_perf)

epdList.append(ThriftEPerf(dataList, "2014-08-13 00:00:00", 60000))

# creating totalIOPS ThriftPerfData object
totalIOPS = ThriftPerfData("totalIOPS", "Total IOPS", epdList)
out['tTotalIOPS'] = totalIOPS

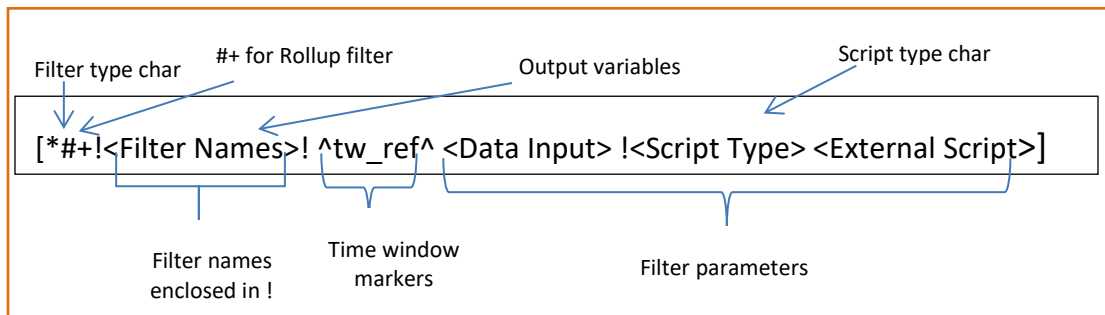
```

## 6.2.2 Rollup external scripts

Rollup external scripts are useful when the requirement is to pull data from related resources and use them to generate results at the parent level resource. Rolling up of data allows the user to analyze data at low granularity.

### 6.2.2.1 Syntax

The syntax of the filter is as follows:



Where:

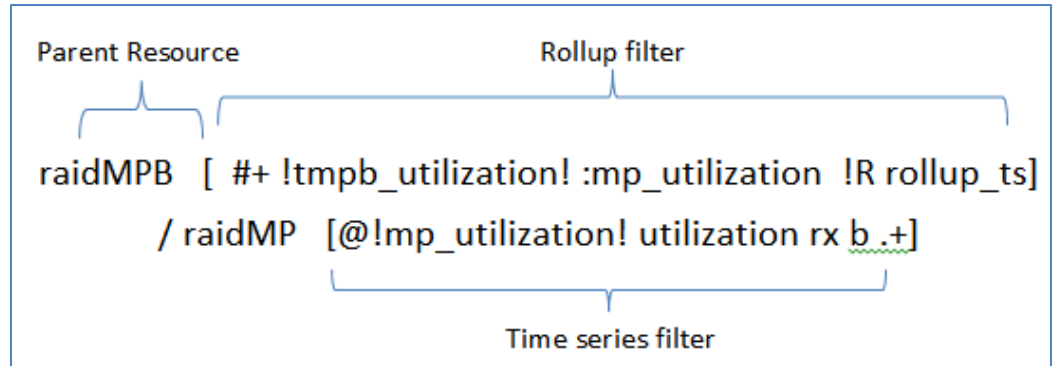
**#+:** The + symbol specifies that it is a rollup filter.

### 6.2.2.2 Use case

Utilization from all related raidMPs needs to be aggregated at the parent MPB resource.

R

Filter



## Input

In the case of a resource rollup, the input RList named as **inputResourceGroup** will consist of multiple lists, each containing the data of a single resource. These individual lists will be named in accordance with the resource they represent, thereby making identification easy.

The following is the structure of a sample list of this kind:

```
List of 2
$ raidMP66297-CL7-J-10-19:List of 3
..$ id : chr "raidMP66297-CL7-J-10-19"
..$ type : chr "raidMP"
..$ mp_utilization:Timeseries:List of 4
.. ..$ unit : chr "percent"
.. ..$ attrId : chr "utilization"
.. ..$ displayName: chr "Utilization"
.. ..$ perfList :List of 3
.. .. ..$ startTime: chr "2014-04-11 05:30:00"
.. .. ..$ interval : num 60000
.. .. ..$ dataList : num [1:80] 5 6 9 7 4 3 3 4 ...
$ raidMP66297-CL7-J-10-20:List of 3
..$ id : chr "raidMP66297-CL7-J-10-20"
..$ type : chr "raidMP"
..$ mp_utilization:Timeseries:List of 4
.. ..$ unit : chr "percent"
.. ..$ attrId : chr "utilization"
.. ..$ displayName: chr "Utilization"
.. ..$ perfList :List of 3
.. .. ..$ startTime: chr "2014-04-11 05:30:00"
.. .. ..$ interval : num 60000
.. .. ..$ dataList : num [1:80] 1.3 0.9 2 3 7 4 7 5 ...
```

RList for each resource would have data corresponding to input variable names specified in the query filter. In addition to the object structure explained in the input section of regular scripts, this list also contains two character objects specifying the resource ID and type. The following table describes the object structure for a resource.

Key	Data type	Description
id	Char	Resource signature (Resource Unique Identifier)
type	Char	Resource type



<attributeId>:Scalar	List	<i>scalarDataObject</i>
<attributeId>:Timeseries	List	<i>timeseriesDataObject</i>

## Processing

The following is a sample R script to add two timeseries data inputs. First, it gets the two timeseries, one each for a resource, out of data inputs and then passes them both to a function named as *function\_to\_add*, which returns a new timeseries. Users can write their own logic to add two timeseries data inputs in function *function\_to\_add*.

```
# The input list consists of two lists, one for each resource.
# The first one contains Utilization metric data for the resource
# raidMP66297-CL7-J-10-19 while the other has data for
# raidMP66297-CL7-J-10-20

# Let's first fetch metric data for the two resources as separate,
# explicit vectors from the input list.
resource1_perf_list <- input_list[[1]]$perfList$dataList
resource2_perf_list <- input_list[[2]]$perfList$dataList

# Now pass these vectors to a function that performs the
# required operation
parent_utilization <- function_to_add(resource1_perf_list, resource2_perf_list)

# Definition of the function called above
function_to_add <- function(vector_1, vector_2) {
  # Code to add the series
}
```

## Output

Rollup script output in R is RList, similar to Regular R script output.

The following is an R script snippet to generate timeseries data output:

```
# Prepare time series
start <- "2014-04-11 05:30:00" # Start time of resultant time series
interval <- 60000 # Interval of time series in seconds
newdata <- rollup_resources(mpb_utilization)
newseries <- list(startTime=start, interval=interval, dataList=newdata)

# List of series
dataVector[[length(dataVector) + 1]] <- newseries

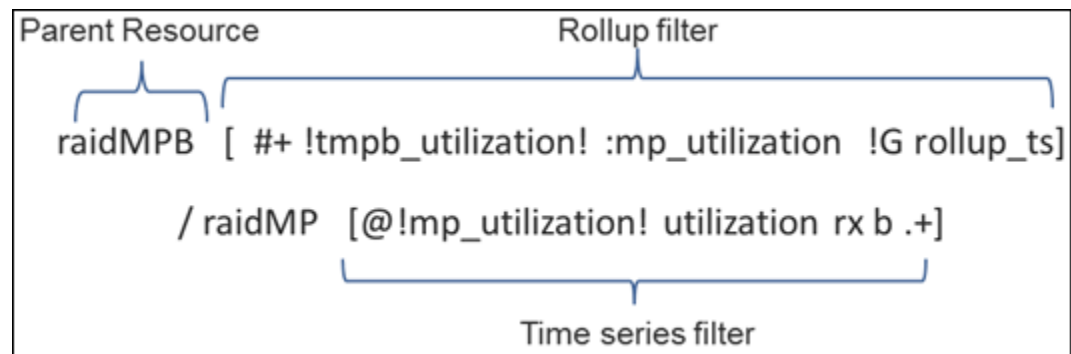
# Create output result
tmpb_utilization[['attrId']] <- 'utilization'
tmpb_utilization[['unit']] <- 'percent'
tmpb_utilization[['displayName']] <- 'Utilization'
tmpb_utilization[['dataType']] <- 'Timeseries'
tmpb_utilization[['data']] <- dataVector
```

The following is the sample output structure for timeseries attribute *utilization*:

```
List of 1
$ attrId      : chr "utilization"
$ unit        : chr "percent"
$ displayName: chr "Utilization"
$ dataType    : chr "Timeseries"
$ data        : List of 1
..$ :List of 3
.. ..$ startTime: chr "2014-04-11 05:30:00"
.. ..$ interval : num 60000
.. ..$ dataList : num [1:80] 4.31 4.35 4.52 4.7 4.64 ...
```

## Groovy

### Filter



### Input

In the case of rollup scripts, the Query engine makes available a special map (inputResourceGroup) to groovy scripts. **inputResourceGroup** holds the list of all the related resource information, where resource information contains the resource signature and the attribute's data specified in the filter.

InputResourceGroup is a map holding resource information corresponding to the resource signature. The following is the object of inputResourceGroup:

```
input: [
  raidMP#66297-CL7-J-10-19:[
    id:raidMP#66297-CL7-J-10-19,
    mp_utilization_Timeseries:[
      unit:percent,
      attrId:utilization,
      perfList:[
        [
          dataList:[5.0, 6.0, 9.0, ..., 3.0, 4.0],
          startTime:2014-04-11 05:30:00,
          interval:60000
        ]
      ],
      displayName:Utilization
    ],
  ],
  raidMP#66297-CL7-J-10-19:[
    id:raidMP#66297-CL7-J-10-19,
    mp_utilization_Timeseries:[
      unit:percent,
      attrId:utilization,
      perfList:[
        [
          dataList:[1.3, 0.9, 2.0, ..., 7.0, 5.0],
          startTime:2014-04-11 05:30:00,
          interval:60000
        ]
      ],
      displayName:Utilization
    ],
  ],
]
```

The object holding resource information is also a map and contains the resource signature corresponding to key 'id' and attribute data corresponding to the attribute ID.

The attribute data object depends on the attribute type, and type is suffixed in the key itself. The following table describes the object structure for a resource.

Key	Data type	Description
id	String	Resource signature (Resource Unique Identifier)
<attributeId>_Scalar	Map	<i>scalarDataObject</i>
<attributeId>_Timeseries	Map	<i>timeseriesDataObject</i>

*scalarDataObject* and *timeseriesDataObject* are already explained for regular script input.

### Processing

The following is a sample groovy script to roll up resource data to the parent level resource. The script iterates over InputResourceGroup to get each resource. Then it gets timeseries out of each resource and performs an operation to consolidate them. Users can write their own logic to consolidate timeseries data inputs.

```
// iterate through inputResourceGroupMap
for ( def id : inputResourceGroup.keySet() ){
    // get each resource
    def resource = inputResourceGroup[id];
    for ( def attr : resource.keySet() ){

        def value = resource[attr];
        // get time series attributes
        if (attr =~ /Timeseries$/ ) {
            // fetch Time series data
            list_of_perf = value.get("perfList");
            // perform operations
        }
    }
}
```

## Output

The following is a sample groovy code to generate output *tmpbUtilization* and add *tmpbUtilization* in outputMap:

```
outputMap = new HashMap<String, Object>();

def perfList = new ArrayList<Map<String, Object>>();

// prepare time series
def dataMap = new HashMap<String, Object>();
dataMap.put("startTime", "2014-04-11 05:30:00"); // start time of resultant time series
dataMap.put("interval", 60000); // interval of time series in milliseconds
dataMap.put("dataList", dataList); // list of double values

perfList.add(dataMap); // list of time series

// create output result map for group.
def rolledUpUtilization = new HashMap<String, Object>();
rolledUpUtilization.put("unit", "percent"); // unit for new attribute
rolledUpUtilization.put("displayName", "MPB Utilization"); // display name for resultant time-series
rolledUpUtilization.put("attrId", "mpbUtilization"); // unique Id for time-series
rolledUpUtilization.put("type", "Timeseries"); // type for resultant time-series
rolledUpUtilization.put("data", perfList); // list of time-series

// put the result in output map against key as in regular filter
outputMap.put("tmpbUtilization", rolledUpUtilization);
```

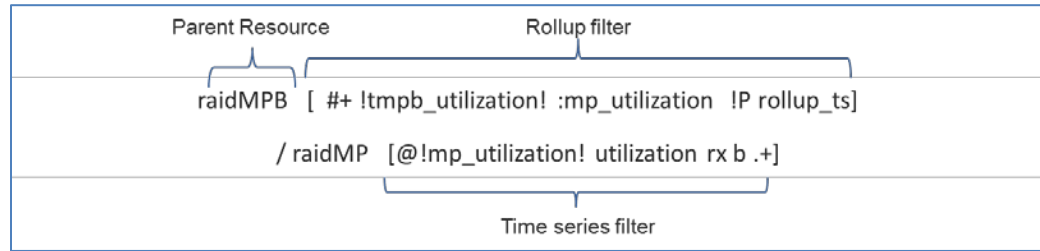
The following is a sample output showing object structure of outputMap having tmpbUtilization:

```
outputMap[
    tmpbUtilization : [ attrId:mpbUtilization,
                        unit:percent,
                        dataType:Timeseries,
                        data : [
                            [ dataList:[4.0, 14.0, ..., 23.0, 16.0],
                              startTime:2014-04-11 05:30:00,
                              interval:60000
                            ],
                            displayName:MPB Utilization
                        ]
    ]
```

Object structure of output data is similar to regular script.

## Python

### Filter



### Input

In the case of rollup script, input to Python script is the list of *ThriftResourceInput*, which is available through *InputResourceGroup*.

*ThriftResourceInput* represents a resource in Python and the following is the object structure:

Keys	Type	Description
signature	String	Unique signature of resource
type	String	Resource type
perfMap	Dictionary<String, ThriftPerfData>	Dictionary that stores timeseries data for an attribute as <i>ThriftEPerfData</i>
doubleScalarMap	Dictionary<String, ThriftDoubleScalar>	Dictionary that stores double scalar data for resource as <i>ThriftDoubleScalar</i>
stringScalarMap	Dictionary<String, ThriftStringScalar>	Dictionary that stores string scalar data for resource as <i>ThriftStringScalar</i>
doubleVectorMap	Dictionary<String, ThriftDoubleVector>	Dictionary that stores double vector data for resource as <i>ThriftDoubleVector</i>
stringVectorMap	Dictionary<String, ThriftStringVector>	Dictionary that stores string vector data for resource as <i>ThriftStringVector</i>

In this example, each MP resource will be depicted as *ThriftResourceInput* in Python script.

### Processing

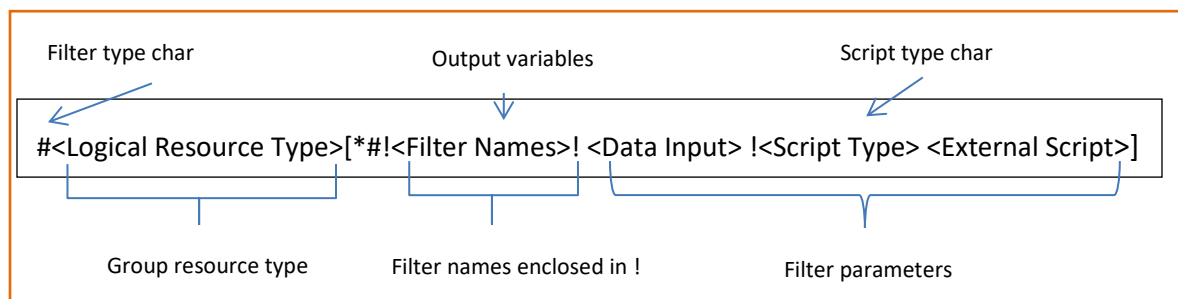
The following is a sample Python script to roll up resource data to the parent level resource. The script iterates over *InputResourceGroup* to get each resource. Then it gets timeseries out of each resource and performs an operation to consolidate them. Users can write their own logic to consolidate timeseries data inputs.

## Output

```
mpbUtilization = ThriftPerfData("percent", "Utilization", epdList)
out['tmpb utilization'] = mpbUtilization
```

### 6.2.3 Group external scripts

### 6.2.3.1 Syntax



- #: Denotes that this is a group filter. Group filter should be the first filter in that level and only one is allowed in a query.

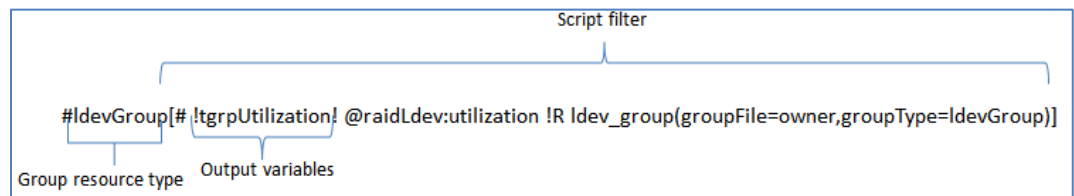
- **Logical Resource Type:** Resource type of the generated group. Alphanumeric and underscores are valid.
- **Filter Names:** Optional. These will be used if this filter generates other outputs in addition to creation of logical group resources.
- **Data Input:** The direct input variables should be preceded with **@** for timeseries data and **=** for scalar data. Data should belong to only one type of resource. The input data variable should be prefixed with resource type (e.g, @h:cpuUsage). The resource signature will be passed implicitly.

### 6.2.3.2 Use case

Create a group of LDEV resources based on their owners and find utilization for each group.

R

#### Filter



#### Input

In case of a group rollup, the input list in R will consist of multiple lists, each containing the data of a single resource. These individual lists will be named in accordance with the resource they represent thereby making identification easy. In addition, a mapping of individual resources to groups may be provided in the form of a file wherein the path to this file will be available in the R environment as an object with the name specified in the query ('groupFile' in the query above).

The section enclosed in '('and ')' is known as additional script inputs and can be provided to regular, rollup, and group scripts wherever applicable. It should be a Key Value pair. Multiple additional scripts can be provided to the script using the separator – ','.

The following is the object structure of a sample list of this type:

```

List of 2
$ raidLdev66297-00^C4^00:List of 3
..$ id : chr "raidLdev66297-00^C4^00"
..$ type : chr "raidLdev"
..$ utilization:Timeseries:List of 4
.. ..$ unit : chr "percent"
.. ..$ attrId : chr "utilization"
.. ..$ displayName: chr "Utilization"
.. ..$ perfList :List of 3
.. .. ..$ startTime: chr "2014-04-11 05:30:00"
.. .. ..$ interval : num 60000
.. .. ..$ dataList : num [1:80] 5 6 9 7 4 3 3 4 ...
$ raidLdev66297-00^C4^01:List of 3
..$ id : chr "raidLdev66297-00^C4^01"
..$ type : chr "raidLdev"
..$ utilization:Timeseries:List of 4
.. ..$ unit : chr "percent"
.. ..$ attrId : chr "utilization"
.. ..$ displayName: chr "Utilization"
.. ..$ perfList :List of 3
.. .. ..$ startTime: chr "2014-04-11 05:30:00"
.. .. ..$ interval : num 60000
.. .. ..$ dataList : num [1:80] 1.3 0.9 2 3 7 4 7 5 ...

```

## Processing

Rolling up using a resource-group mapping can be done in two steps. In the first step, either define the mapping manually in the R environment or read it from a file whose path is available in the variable 'groupFile'. R provides a construct called *factor* that allows this kind of grouping and makes it convenient to perform operations across these groups.

Once the resources are split by group, pass resources of each group to a function that aggregates input data for them.



```

# Mapping of an individual resource to a group can be
# made available in the form of a file. This file can
# be in any standard format like .txt or .csv and
# depending on the file type, logic for associating
# each resource to a group will need to be modified.

# At its very core, these factors can be created using
# the 'factor' construct wherein every resource name is
# associated to a factor level (i.e. a group). These factor
# levels can then be used to segregate and aggregate the
# data as required.

# Create a ldev to Owner mapping
ldev_factor <- apply_resource_group(groupFile)

# Definition of mapping function called above
apply_resource_group <- function(map_filepath) {
  # Code to associate every resource to a group
}

# In this example, consider that every ldev is mapped to one
# of two possible owners.
rollup_group1 <- rollup_resources(input_list[[ldev_factor[[1]]]])
rollup_group2 <- rollup_resources(input_list[[ldev_factor[[2]]]])

# Definition of rollup function called above
rollup_function <- function(ldev_list) {
  # Code to aggregate grouped resource metrics as required
}

```

## Output

Output for the group script filter is an RList named 'outputResourceGroup' comprised of logical resources. These logical resources are RList having ID, type, list of constituting resources, and other timeseries, scalar, or vector output created by the script.

The following is a sample R pseudo-code to create logical resources and associated data from input resource data.

```
tgrpUtilization <- list(
  list(
    "id"="ldevGroup1",
    "resourceList"=ldevnames_bygroup[[1]],
    "tgrpUtilization"=list(
      "attrId"="groupUtilization",
      "unit"="percent",
      "displayName"="ldevGroup1",
      "type"="Timeseries",
      "data"=list(
        "startTime"="2014-04-11 05:30:00",
        "interval"=60000,
        "dataList"=rollup_group1)),
    "type"="ldevGroup"
  ),
  list(
    "id"="ldevGroup2",
    "resourceList"=ldevnames_bygroup[[2]],
    "tgrpUtilization"=list(
      "attrId"="groupUtilization",
      "unit"="percent",
      "displayName"="ldevGroup2",
      "type"="Timeseries",
      "data"=list(
        "startTime"="2014-04-11 05:30:00",
        "interval"=60000,
        "dataList"=rollup_group2)),
    "type"="ldevGroup"
  )
)
```

Key	Data type	Description
id	Char	Unique identifier for logical resource
resourceList	List	List of related resource signatures
<output variable name>	List	Attribute output data as explained in regular script output.
type	Char	Resource type of logical group

The following is a sample output for a group script:

```

List of 2
$ :List of 4
..$ id : chr "ldevGroup1"
..$ resourceList : chr [1:3] "raidLdev66297-00^CL4^00" "raidLdev66297-00^CL4^01" "raidLdev66297-00^CL4^04"
..$ tgrpUtilization:List of 5
.. ..$ attrId : chr "groupUtilization"
.. ..$ unit : chr "percent"
.. ..$ displayName: chr "ldevGroup1"
.. ..$ type : chr "Timeseries"
.. ..$ data :List of 3
.. .. ..$ startTime: chr "2014-04-11 05:30:00"
.. .. ..$ interval : num 60000
.. .. ..$ dataList : num [1:80] 4.9 4.81 4.78 5.37 2.15 ...
..$ type : chr "ldevGroup"
$ :List of 4
..$ id : chr "ldevGroup2"
..$ resourceList : chr [1:2] "raidLdev66297-00^CL4^02" "raidLdev66297-00^CL4^03"
..$ tgrpUtilization:List of 5
.. ..$ attrId : chr "groupUtilization"
.. ..$ unit : chr "percent"
.. ..$ displayName: chr "ldevGroup2"
.. ..$ type : chr "Timeseries"
.. ..$ data :List of 3
.. .. ..$ startTime: chr "2014-04-11 05:30:00"
.. .. ..$ interval : num 60000
.. .. ..$ dataList : num [1:80] 4.86 6.68 3.62 4.66 5.97 ...
..$ type : chr "ldevGroup"

```

## Groovy

### Filter



### Input

In case of a group rollout, the input in groovy is a map 'InputResourceGroup'. inputResourceGroup contains multiple resources corresponding to the key having its resource signature. In addition, a mapping of individual resources to groups can be provided in the form of a file wherein the path to this file will be available in the groovy environment as an object with the name specified in the query ('groupFile' in the query above).

Each resource holds resource signature as *id* and attribute's data corresponding to input variable name (attributeId\_type). Object structure of **inputResourceGroup** is the same as rollout script inputs.

```

InputResourceGroup: [
  raidLdev#66297-00^C4^00 :[
    id:raidLdev#66297-00^C4^00,
    utilization_Timeseries :[
      unit:Percent,
      attrId:utilization,
      perfList :[
        [
          dataList:[0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
          startTime:2014-04-11 05:30:00,
          interval:60000
        ]
      ],
      displayName:Utilization
    ],
  ],
]

```

## Processing

Rolling up using a resource-group mapping can be done in two steps. In the first step, either define the mapping manually in the groovy environment or read it from a file whose path is available in the variable 'groupFile'. The script provides mapping in the form of a map, which is used to aggregate resources across these groups.

Once the resources are split by group, pass resources of each group to a function that aggregates timeseries for them.

```
/*
 * create ldev to owner mapping
 * raid ldev -> owner
 */
def ldevToOwnerMapping = createLdevToOwnerMapping();

/*
 * combine multiple ldev's data into groups using ldevToOwnerMapping.
 * owner name -> resource Id -> resource
 */
def ownerData = combineLdevDataInToGroups(ldevToOwnerMapping);

// iterate over each owner and create owner output map
for (String ownerName : ownerData.keySet()) {
    def owner = ownerData.get(ownerName);

    // create owner output map for each group
    def ownerOutputMap = createOwnerOutputMap(owner);

    // Adds owner output map into OutputResourceGroup
    addOwnerData(ownerOutputMap, ownerName, owner)
}
```

## Output

The output list structure for group script is slightly different. A special list, *outputResourceGroup*, needs to be created in groovy script to return output. *outputResourceGroup* contains all group resources and each resource consists of *id*, *type*, *resourceList*, and the attribute's data.

The following is a sample groovy script to prepare owner utilization and to add owner utilization into *outputResourceGroup*.

The script prepares group utilization and adds the group resource into *resMap*. The group resource contains *id*, *type*, *tgrputilization*, and *resourceList* containing related resources. It adds *resMap* into *outputResourceGroup*.

```
// create output perf List
def perfList = new ArrayList<Map<String,Object>>();

def dataMap = new HashMap<String,Object>();
dataMap.put("startTime", df.format(startTime)); // start time of resultant time series
dataMap.put("interval", groupInterval); // interval of time series in milliseconds
dataMap.put("dataList", _ownerOutputMap.values()); // list of double values

perfList.add(dataMap); // list of time series

// create output result map for group.
def grpUtilizationMap = new HashMap<String,Object>();
grpUtilizationMap.put("unit", groupUnit); // unit for new attribute
grpUtilizationMap.put("displayName", _ownerName); // display name for resultant time-series
grpUtilizationMap.put("attrId", "groupUtilization"); // unique Id for time-series
grpUtilizationMap.put("type", "Timeseries"); // type for resultant time-series
grpUtilizationMap.put("dataType", "Double"); // data type of resultant time-series
grpUtilizationMap.put("data", perfList); // list of time-series

// create resource map for group
def resMap = new HashMap<String, Object>();
resMap.put("id", _ownerName); // unique group name
resMap.put("type", groupType); // group type
resMap.put("tgrpUtilization", grpUtilizationMap); // output time-series map
resMap.put("resourceList", _owner.keySet()); // list of resources belonging to this group

// insert resource map into output resource list
outputResourceGroup.add(resMap);
```

The following is a sample output of outputResourceGroup:

```
OutputResourceGroup::[
  [
    id:ldevGroup2,
    resourceList:[
      raidLdev#66297-00^00^C4,
      raidLdev#66297-00^C4^02
    ],
    tgrpUtilization:[
      attrId:groupUtilization,
      unit:Percent,
      dataType:Double,
      data:[
        [
          dataList:[0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
          startTime:2014-04-11 05:30:00,
          interval:60000
        ]
      ],
      type:Timeseries,
      displayName:ldevGroup2
    ],
    type:ldevGroup
  ],
]
```

outputResourceGroup is a list containing group resources and each group resource is also represented as a map. The following is the object structure of a resource:

Key	Data type	Description
id	String	Unique identifier for logical resource
resourceList	List	List of related resource signatures

Key	Data type	Description
<output variable name>	Map	Attribute output data. It is explained in regular script output.
type	String	Resource type of logical group

## Python

### Filter



### Input

In case of group script, input to Python script is the list of *ThriftResourceInput*. Object structure for *ThriftResourceInput* is already explained in the rollup script.

In this example, Each *ThriftResourceInput* Object in input represents one LDEV object.

### Processing

Rolling up using a resource-group mapping can be done in two steps. In the first step, either define the mapping manually in the Python environment or read it from a file whose path is available in the variable 'groupFile'.

Once the resources are split by group, script aggregates timeseries of all resources belonging to one group.

```
#implement execute method
def execute(**kwargs) :
    # fetch inputResourceGroup List
    inputResourceGroupList = kwargs['inputResourceGroup']

    outputResourceGroup = []

    #iterate over each object
    for resource in inputResourceGroupList :
        # fetch utilization of each series and group them as per requirement
        # Provide type and Name for Resource
        # Additionally create any timeseries scalar or vector data associated with them.
        resGroup = ['raidLdev#1','raidLdev#2']
        outputResourceGroup.append(ThriftResourceGroup(type, name, resGroup, {
            'tgrpUtilization':ThriftPerfData(...), {}, {}, {}, {}))

    #create final output
    return out
```

### Output

Below is a sample Python script to create a group resource and to add the logical resource in the out dictionary.

```

resGroup = ['raidLdev#1','raidLdev#2']
outputResourceGroup.append(ThriftResourceGroup(type, name, resGroup, {
'tgrpUtilization':ThriftPerfData(...)}, {}, {}, {}, {}))
.....
out['outputResourceGroup'] = outputResourceGroup
#create final output
return out

```

For logical grouping, Python script is expected to return the list of *ThriftResourceGroup* objects. Each *ThriftResourceGroup* object represents a logical resource created in Python where name, type, and list of contained resources are mandatory.

Object structure of *ThriftResourceGroup*:

Key	Type	Description
Type	String	Logical resource type
name	String	Logical resource name
resourceSignatureList	List<String>	List of resource signatures in Logical Resource
perfMap*	Dictionary<String, ThriftPerfData>	attributeld -> ThriftEPerfData objects to store timeseries data for resource
doubleScalarMap*	Dictionary<String, ThriftDoubleScalar>	attributeld -> ThriftDoubleScalar objects to store double scalar data for resource
stringScalarMap*	Dictionary<String, ThriftStringScalar>	attributeld -> ThriftStringScalar objects to store string scalar data for resource
doubleVectorMap*	Dictionary<String, ThriftDoubleVector>	attributeld -> ThriftDoubleVector objects to store double vector data for resource
stringVectorMap*	Dictionary<String, ThriftStringVector>	attributeld -> ThriftStringVector objects to store string vector data for resource

\* represents optional parameter

# 7 Advanced features

## 7.1 Query Join

This feature allows MQL to relate disjoint resources by allowing the user to query resources, which do not have relation specified in the DB usually based on some scalar attribute.

Users may want to correlate few resources and view the entire or part of the infrastructure. However, data or resource to compose this view might come from different data subsets, and there might be cases where these resources from different data subsets do not have relation specified amongst them. In this case, these resources would be disjoint.

However, logically such resources can be related to each other in the real world but due to lack of relation specification in the Analyzer detail view DB, the query won't be able to fetch these resources. The Query Join feature overcomes this limitation by allowing the user to query resources, which do not have relation specified in the DB.

### 7.1.1 Syntax

```
R1/⚠R2
```

Where R1 and R2 do not have direct relation in the Analyzer detail view DB.

In this query, all R2 resources will be returned as related resource for each R1 resource.

**\*\*This is not a general use-case.**

```
R1[=!ref!scalarAttr1 rx <regex>]/^R2[=scalarAttr2 rx :ref]
```

Where '**R2**' is unrelated to '**R1**'.

**R2** will be filtered for having same *scalarAttr2* as **R1**'s *scalarAttr1* attribute.



The following table describes query filter options applicable to Query Join:

Query filter options	Descriptions	Valid values
trType	Specifies the transformation type that the matching pattern would undergo before application	c2c/r2s where, c2c is character to character transformation r2s is regex to String transformation.
trFrom	In case of <b>trType c2c</b> , it represents a list of characters, which will be replaced by corresponding character in <b>trTo</b> parameter or null (if no corresponding character is found)  In case of <b>trType r2s</b> , it represents a regex pattern to be replaced by string in <b>trTo</b> for all occurrences in input.  It should be enclosed in double quotes ("").	List of characters, if trType = c2c
		Regex Pattern, if trType = r2s
trTo	In case of <b>trType c2c</b> , it represents a list of characters with which corresponding character of <b>trFrom</b> will be replaced. It is an optional parameter. In this case, the characters in <b>trFrom</b> will be completely removed from input.  In case of <b>trType r2s</b> , it represents a string, which replaces all the occurrences of text matching regex specified in <b>trFrom</b> .  It should be enclosed in double quotes ("").	List of characters, if trType = c2c.
		Text string, if trType=r2s

In addition to basic filtering based on scalar attribute regex matching, some extra processing may be required. Here are few such syntaxes.

It is possible that values for scalar attributes at R1 and R2 have similar but not exactly same values. For example, the case (uppercase/lowercase letters) of these values can be different. In this scenario, we need to compare while ignoring case. This can be achieved with the following query syntax.

```
R1[=!ref!scalarAttr1 rx <regex>]/^R2[=scalarAttr2 rx (?i) :ref]
```

The value of scalarAttr2 for R2 will undergo transformation before being matched with the scalar attribute value from R1.

**trType=c2c** specifies that the scalarAttr2 value needs to go with character to character transformation. This can be achieved with the following query syntax.

```
R1[=!ref!scalarAttr1 rx <regex>]/^R2[=scalarAttr2 rx :ref] {  
trType=c2c,trFrom=":",trTo="."}
```

**trFrom** can take a list of characters and replace each one of them with the corresponding index character from **trTo** before matching. In above example ``:`` will be replaced by ``.`' before matching scalarAttr2 value with scalarAttr1.

If no corresponding character is found for replacement in the target then the character is replaced by null.

```
R1[=!ref!scalarAttr1 rx <regex>]/^R2[=scalarAttr2 rx :ref] {  
trType=r2s,trFrom="netApp_",trTo="x"}
```

Another value for **trType** is **r2s**, in which case a regex is specified in **trFrom** and that is replaced by the string mentioned in **trTo** before matching.

## 7.1.2Use case

The Analyzer detail view database may have switch and storage data, which are not related explicitly in the DB. However, they may be related in the real world. Therefore, to find this relation, you can use scalar attribute values from resources of these two data subsets.

Let's say the WWN value from RaidPort from storage and fabCiscoEndPort of switch are the same.

- Then the user may exploit the commonality of this value to deduce a relationship between switch and storage data as shown in following query:

```
fabCiscoEndPort[=!ref!wwn rx .*/^raidPort[=wwn rx :ref]
```

This query will return all raidPorts having the same WWN value as of end port as its related resource.

- It might also be possible that values of scalar attribute are not exactly the same. However similar, in such case, the user may use filter options to modify values to match before they are matched.

For example:

```
fabCiscoEndPort[=!ref!wwn rx .*/^raidPort[=wwn rx  
:ref]{trType=c2c,trFrom=".",trTo=":"}
```

The above query will match the raidPort WWN value with the endPort WWN value after substituting . (dot) with : (colon) in the endPort WWN value.

- The user may also use regex replacement if required, as displayed in following example:

```
fabCiscoEndPort[=!ref!wwn rx .*]/^raidPort[=wwn rx
:ref]{trType=r2s,trFrom="\s",trTo=":"}
```

This query will replace all white space with colon from the end port WWN value before matching it with the RAID port's WWN value.

## 7.2 Synthetic attribute

Synthetic attributes are computed/synthesized on the server by aggregating data from other attributes present in the server DB.

Synthetic Attributes do not have their corresponding data stored directly in the server DB. Their value is computed by applying aggregations on data present for other metrics in the server DB.

Synthetic Attribute definitions are defined in AttributeDef.xml. When queried for such metric, MQL will expand internally using the definition specified in XML to aggregate, and produce the expected result.

### 7.2.1 Syntax

#### 7.2.1.1 Definition

One can define Synthetic Attribute in AttributeDef.xml as in the following example:

```
<AttributeDef id="<newAttrId>" name="<Display Name>" type="timeseries" unit="<unit>"
synthetic="true" outInterval="<outputInterval>" inputInterval="<input Interval>"
intervalRollupOp=<op> resourceRollupOp="<op>" derivedAttrOp="<op>" counterList="<input
counter list>" resPath="<relation path>"/>
```

Where,

XML Attribute	Description
id	Unique identifier for the attribute
name	Display name for the attribute
type	Attribute type, valid values are scalar and timeseries. However, currently synthetic attributes are supported only for the timeseries attribute.
unit	Unit for the attribute
synthetic	This XML attribute should be true to specify that this definition is for synthetic attributes
outInterval	Parameter to specify the Output timeseries interval, in seconds; it should be used for interval rollup

XML Attribute	Description
inputInterval	Optional parameter to specify if any particular interval data layer should be used for aggregations
intervalRollupOp	Parameter to specify which operation should be used for interval rollup , valid values are SUM/AVG/MIN/MAX
resourceRollupOp	Parameter to specify which operation should be used for resource rollup; valid values are SUM/AVG/MIN/MAX/weightedAvg
derivedAttrOp	Parameter to specify which operation should be used for derived attribute; valid values are SUM/AVG/DIFF/weightedAvg
counterList	Comma-separated list of counters that are collected directly by probe on which operations need to be performed to get the desired output
resPath	Optional parameter to specify the relational path (with filters, if required) to the resource at which counters specified in counterList are available.

### 7.2.1.2 Usage

The user can use the Synthetic Attribute as any other static attribute definition in a query. For example:

```
R1[@<synAttrId> rx b .*]
```

The above query when executed will look up the synAttrId definition and will then internally expand to evaluate the metric value based on configurations specified in its definition.

### 7.2.2 Use case

#### Synthetic Attribute for interval rollup

```
AttributeDef id="cpuUsage_rolledUp" name="CPU Usage - 1 hr"
type="timeseries" unit="Percent" synthetic="true" outInterval="3600"
inputInterval="60" intervalRollupOp="AVG" counterList="cpuUsage"/>
```

#### Synthetic Attribute for derived attribute

```
AttributeDef id="writeIOPS" name="Write IOPS" type="timeseries" unit="KBps"
synthetic="true" derivedAttrOp="DIFF" counterList="totalIOPS,readIOPS"/>
```

#### Synthetic Attribute for resource rollup

```
AttributeDef id="totalIOPS" name="Total IOPS" type="timeseries" unit="KBps"
synthetic="true" resourceRollupOp="SUM" counterList="totalIOPS"
resPath="raidLdev"/>
```

#### Synthetic Attribute for interval rollup + derived attribute + resource rollup

```
AttributeDef id="totalIOPS" name="Total IOPS" type="timeseries" unit="KBps"
synthetic="true" derivedAttrOp="SUM" outInterval="3600"
intervalRollupOp="SUM" resourceRollupOp="SUM"
counterList="writeIOPS,readIOPS" resPath="raidLdev"/>
```

## 8 Appendix

1. To execute R scripts from MQL, there are certain prerequisites:

- R must be installed on Host system.
- Rserve instances should be running either locally or remotely on an accessible host.
- Rserve Library should be installed on Host where R server needs to be running. User can start Rserve instance locally using this command:

```
Rserve(port=6311, args="--no-save --RS-conf
/usr/local/megha/ExternalScript/R/Rserv.cfg")
```

- Following properties need to be configured in `ext.script.properties`:
    - `ext.script.R.Home` – set to the installed location of R on host machine.
    - `ext.script.R.server.count` – number of R servers running to process MQL Requests.
    - `ext.script.R.<count>.hostname` – host on which R server is running.
    - `ext.script.R.<count>.port` – port on which R server is listening at `ext.script.R.<count>.hostname` Host.
2. Scripts are executed on Groovy interfaces and certain such interfaces are pooled in Analyzer detail view for faster execution. Number of pooled Groovy interfaces can be configured from the property '`ext.script.groovy.instance.count`' in `ext.script.properties` under the `conf/sys` folder.

Since the groovy script runs on JVM, if the number of parallel requests for groovy script execution increases the limit of the pooled groovy interface count, a new groovy interface is spawned. However, these new interfaces are not added back to the interface pool.

3. To use Python scripts from MQL, there are certain prerequisites:
  - Python should be installed on Host.
  - Thrift should be installed on Host. This can be done as explained in the following steps.
    - `pushd /usr/local/megha/ExternalScript/Thrift/`
    - `unzip thrift.zip`

- pushd *./thrift\*/*
- python *setup.py install*
- Popd
- Popd
- Python Thrift server should be running on ports and interfaces that should be configured in *ext.script.properties*
  - Starting Python server:

```
python
/usr/local/megha/ExternalScript/Thrift/py_impl/PythonS
erver.py <ipaddr> <port>
```
- Configure setting in *ext.script.properties*:
  - Set *ext.script.P.server.count*, i.e., number of Python Thrift servers available to process request.
  - Set hostname and port for all Python servers by setting properties:
    - *ext.script.P.<count>.hostname* – IP Address of host where Thrift server is running
  - *ext.script.P.<count>.port*- Port on which Thrift server is listening for request on specified (*ext.script.P.<count>.hostname*) host.

## Hitachi Vantara Corporation

Corporate Headquarters  
2845 Lafayette Street  
Santa Clara, CA 95050-2639 USA  
[www.HitachiVantara.com](http://www.HitachiVantara.com) | [community.HitachiVantara.com](http://community.HitachiVantara.com)

Regional Contact Information  
Americas: +1 866 374 5822 or [info@hitachivantara.com](mailto:info@hitachivantara.com)  
Europe, Middle East, and Africa: +44 (0) 1753 618000 or [info@emea@hitachivantara.com](mailto:info@emea@hitachivantara.com)  
Asia Pacific: + 852 3189 7900 or [info.marketing.apac@hitachivantara.com](mailto:info.marketing.apac@hitachivantara.com)