Filter

A filter examine an event and decides if it matches or doesn't. Filters are mainly used in rules to decide if the processing entries should be executed on the given event. They're also used for settings, and systems like the Graph Correlator re-use Highland Park's filter mechanism to specify which alarms fit in a correlation. Some publishers may produce topics with a lot of volume and a subscriber may want only a portion of those messages. The subscriber can certainly filter messages after receiving them, but it may be substantially more efficient to ask the API server to filter the messages before sending them to the subscriber.The standard library includes a number of simple filters. The Equals filter, for example, compares a field's value with some other value and returns true if the values match. The standard library also includes filter classes called And, Or, and Not, so you can compose more complex filters. For example, written in the standard JSON config format:

```
"filter":{
"class":"And",
"filters":
[
{ "class":"Equals", "foo":"abc" },
{ "class":"Assigned", "field":"bar" }
]
}
```

This filter matches events in which the field "foo" has value "abc" and the field "bar" is assigned to some value (as opposed to not being present on the event). Filters are used by the consumers to filter out data and consume only specific set of data which matches the conditions mentioned in filter. Filters can be passed as a query parameter by the consumer in consume request as mentioned below:
http://localhost:8080/DMaaP/dmaaprest/events/DMaaP/consumergroup/mHOeNFY4XiWx4CBa?filter=\{"class":"Equals", "field":"email", "value":"test@abc.com" }
Filters can be applied only on data in JSON format i.e. if applied, filters will automatically ignore any non-json data. While consuming, request CONTENT_TYPE is not relevant to filter.

All the supported filter can be found below.

Types of Filters

DMaaP Message Router supports all the filters which were supported by DMaaP Message Router and are mentioned below:-
All Alarms:
Match all alarms.
And:
Create a set of filters. This filter matches when all of them matches.

Field	Description	Type	Notes
filters	Combined Filters	LIST	A list of filters

Assigned:
Choose a field from the event to check for assignment. This filter matches when the field is assigned.

Field	Description	Type	Notes
field	The field to check for on the event.	STRING	A field name
emptyIsAssigned	If true, an empty value is considered an assignment.	BOOLEAN	True or False

Contains:
Check if a search string contains another string.

Field	Description	Type	Notes
String	The value to search. Supports ${} notation.	STRING	Any string
Value	The value to search for. Supports ${} notation.	STRING	Any string

EndsWith:
Check if a search string ends with another string.

Field	Description	Type	Notes
string	The value to search. Supports ${} notation.	STRING	Any string
value	The value to search for. Supports ${} notation.	STRING	Any string

Equals:
Choose a field from the event and a value to check for equality.

Field	Description	Type	Notes

field	The field to check. Supports ${} notation.	STRING	Any string
value	The value to match. Supports ${} notation.	STRING	Any string

FlatironObjectExists
Matches when the given object exists in the given Flatiron instance.

Field	Description	Type	Notes
oid	The OID of the object to look for.	STRING	Any string
flatiron	The name of the Flatiron client instance.	STRING	Any string

IsAging
Choose a field to test. This filter matches if the expression is numeric.

Field	Description	Type	Notes
field	The field to test. Supports ${} notation.	STRING	Any string

IsNumeric
Choose a field to test. This filter matches if the expression is numeric.

Field	Description	Type	Notes
field	The field to test. Supports ${} notation.	STRING	Any string

MathCondition
Choose a field from the event and a value for logical math conditions.

Field	Description	Type	Notes
Field	The field to check. Supports ${} notation.	STRING	Any string
Value	The value to consider. Supports ${} notation.	STRING	Any string
operator	The operation.	STRING	One of { "<=", ">=", ">", "<" }

NoAlarms
Don't match any alarms.
Not
Negate the configured filter.

Field	Description	Type	Notes
filter	The filter to negate.	FILTER	A filter

NotEqual
Choose a field from the event and a value to check for inequality.

Field	Description	Type	Notes
field	The field to check. Supports ${} notation.	STRING	Any string
value	The value to match. Supports ${} notation.	STRING	Any string

NotOneOf
Match when the specified field does not have a value from the given list.

Field	Description	Type	Notes
field	The field to test. Supports ${} notation.	STRING	Any string
values	The matching values.	LIST	A list of strings

OneOf
Match when the specified field has a value from the given list.

Field	Description	Type	Notes
field	The field to test. Supports ${} notation.	STRING	Any string
values	The matching values.	LIST	A list of strings

Or
Create a set of filters. This filter matches when any one of them matches.

Field	Description	Type	Notes
filters	Combined Filters	LIST	A list of filters

RegEx
Choose a field from the event to match against the regular expression you provide.

Field	Description	Type	Notes
field	The text to check for a match. Supports ${} notation.	STRING	Any string
value	The regular expression (pattern) to match.	STRING	Any string

StartsWith
Check if a search string starts with another string.

Field	Description	Type	Notes
string	The value to search. Supports ${} notation.	STRING	Any string
Value	The value to search for. Supports ${} notation.	STRING	Any string

Unassigned
Choose a field from the event to check for assignment. This filter matches when the field is not assigned.

Field	Description	Type	Notes
field	The field to check for on the event.	STRING	A field name
emptyIsAssigned	If true, an empty value is considered an assignment.	BOOLEAN	True or False

WithinSecondsFrom
This filter matches when the specified epoch time value is within the given number of seconds from the baseline time value. Both time values are assumed to be in seconds. If a value is in milliseconds, set baselineTimeInMillis and/or eventTimeInMillis to true.

Field	Description	Type	Notes
field	The time value to test. Supports ${}	STRING	A field name
eventTimeInMillis	Whether to convert the event value from milliseconds.	BOOLEAN	True or False

seconds	The number of seconds.	NUMBER	A number
baselineTimeInMillis	Whether to convert the baseline value from milliseconds.	BOOLEAN	True or False
baseline	The baseline time value. Supports ${}.	STRING	Any string

WithinTimeFromNow
This filter matches when the named field has an epoch time value within the given number of seconds from the current time. The event's time value is assumed to be in seconds. If it's in milliseconds, set eventTimeInMillis to true.

Field	**Description**	**Type**	**Notes**
field	The field to check on the event.	STRING	A field name
eventTimeInMillis	Whether to convert the event value from milliseconds.	BOOLEAN	True or False
seconds	The number of seconds.	NUMBER	A number

Limit:

- Limit is the integer value and DMaaP Message Router will consumes only that set of message which are specified in limit.

Suppose if we set limit=2, then only 2 sets of data will be consumed.
Get ***http://localhost /DMaaP/dmaaprest/events/<<topicName>>/group/2?limit=4***
Let us suppose if
No of data available = 4
Set limit = 6
i.e. limit>no of data
In this scenario all 4 sets of data will be consumed.

- If limit is not passed with the url then by default limit is set to 4096.

i.e. 4096 sets of data will be consumed.
Timeout and Long Poll:

- Timeout is the integer value which will be treated by DMaaP Message Router as time in millisecond.

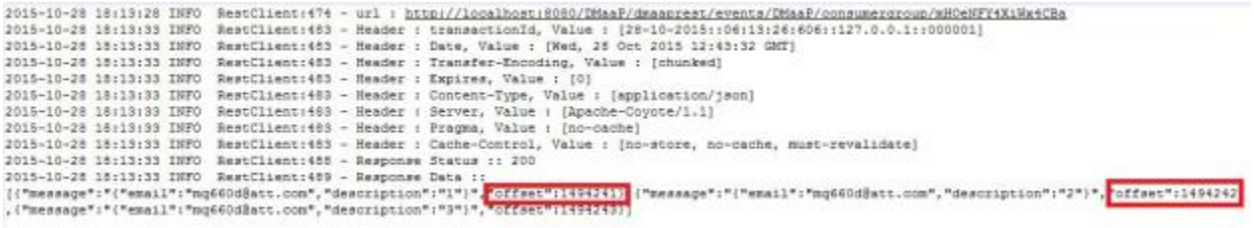
- Get

http://localhost/DMaaP/dmaaprest/events/<<topicName>>/group/2?timeout=20000

- If there is no data available to be consumed, then DMaaP Message Router will poll for the particular period of time specified in timeout this mechanism is known as Long Poll.
- If timeout is not passed with url then DMaaP Message Router will set the value of timeout =10000
- i.e. if no set of data are available then DMaaP Message Router will poll for 10000 ms.

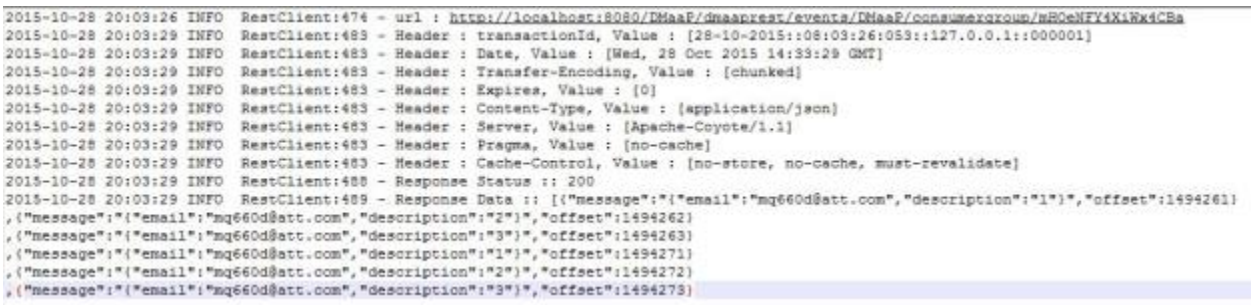
Meta:

- Meta is a Boolean value.
- DMaaP Message Router reads the value of meta from MRConfiguration.properties file at the time of startup.
- If the value of meta is not null and if value of meta is one of these values true, yes, on, 1, y, checked then DMaaP Message Router will take meta flag as true, else it will be false.
- If meta is set to true then consumer will get the value of message offset along with message.



Pretty:

- Pretty is a Boolean value.
- DMaaP Message Router reads the value of pretty from MRConfiguration.properties file at the time of startup.
- If the value of pretty is not null and if value of pretty is one of these values true, yes, on, 1, y, checked then DMaaP Message Router will take pretty flag as true, else it will be false.
- If pretty is set to true then different sets of messages will be printed in next line separated by comma (,).



Filter

A filter examine an event and decides if it matches or doesn't.
Filters are mainly used in rules to decide if the processing entries should be executed on the given event. They're also used for settings, and systems like the Graph Correlator re-use Highland Park's filter mechanism to specify which alarms fit in a correlation.
The standard library includes a number of simple filters. The Equals filter, for example, compares a field's value with some other value and returns true if the values match.
The standard library also includes filter classes called And, Or, and Not, so you can compose more complex filters. For example, written in the standard JSON config format:

```
"filter":{
"class":"And",
"filters":
[
{ "class":"Equals", "foo":"abc" },
{ "class":"Assigned", "field":"bar" }
]
}
```

This filter matches events in which the field "foo" has value "abc" and the field "bar" is assigned to some value (as opposed to not being present on the event).
Filters are used by the consumers to filter out data and consume only specific set of data which matches the conditions mentioned in filter.
Filters can be passed as a query parameter by the consumer in consume request as mentioned below:
http://localhost:8080/DMaaP/dmaaprest/events/DMaaP/consumergroup/mHOeNFY4XiWx4CBa?filter=\{"class":"Equals", "field":"email", "value":"test@abc.com" }
Filters can be applied only on data in JSON format i.e. if applied, filters will automatically ignore any non-json data.
While consuming, request CONTENT_TYPE is not relevant to filter.

The MR API allows a subscriber pass a Highland Park filter as part of the GET request. This will filter the stream of messages sent back to the subscriber, but for this to work, there are some requirements:

- The message payload must be JSON
- Only a filter built from Highland Park's Standard Library may be used. (The Cambria API server doesn't have access to plugged in filters.)
- The filter must be encoded properly in the URL path.

Server-side filtering can also be setup in the Java client as illustrated below

Filtering Consumer
You can also provide a Highland Park filter to your consumer instance, and this filter is passed on to the server in the GET request. One way to create the filter is programmatically. In your code, instantiate a filter from the Highland Park Standard Library Then create a String representation of the filter using the FilterIo.write utility. This String can then be passed to the Cambria client instance for use on the server.

Remember, only Highland Park standard library filter components can be used -- no plug-ins are available in the Cambria server context.

```
package org.onap.sa.highlandPark.integration;

import java.io.IOException;

import java.util.UUID;


import org.onap.nsa.cambria.client.CambriaClientFactory;

import org.onap.nsa.cambria.client.CambriaConsumer;

import org.onap.sa.highlandPark.processor.HpEvent;

import org.onap.sa.highlandPark.stdlib.filters.FilterIo;

import org.onap.sa.highlandPark.stdlib.filters.OneOf;


public class ExampleFilteringConsumer

{

          public static void main ( String[] args ) throws IOException, InterruptedException

          {

                    // Cambria clients take a set of 1 or more servers to use in round-robin
fashion.

                    // If a server becomes unreachable, another in the group is used.

                    final String
serverGroup="ueb01hydc.it.att.com,ueb02hydc.it.att.com,ueb03hydc.it.att.com";


                    // choose a topic

                    final String topic = "TEST-TOPIC";


                    // Cambria clients can run in a cooperative group to handle high-volume topics.

                    // Here, we create a random group name, which means this client is not re-
startable.

                    final String consumerGroup = UUID.randomUUID ().toString ();

                    final String consumerId = "0";


                    // Cambria clients can sit in a tight loop on the client side, using a long-poll

                    // to wait for messages, and a limit to tell the server the most to send at a
time.

                    final int longPollMs = 30*1000;

                    final int limit = -1;


                    // The Cambria server can filter the returned message stream using filters from
the
```

```
                    // Highland Park system. Here, we create a simple filter to test for the AlarmID
                    // value being one of the Mobility power alarms.
                    final OneOf oneOf = new OneOf ( "AlarmId", kPowerAlarms );

                    // create the consumer
                    final CambriaConsumer cc = CambriaClientFactory.createConsumer ( serverGroup,
topic,
                              consumerGroup, consumerId, longPollMs, limit, FilterIo.write ( oneOf )
);

                    // now loop reading messages. Note that cc.fetch() will wait in its HTTP receive
                    // method for up to 30 seconds (longPollMs) when nothing's available at the
server.
                    long count = 0;
                    while ( true )
                    {
                              for ( String msg : cc.fetch () )
                              {
                                        System.out.println ( "" + (++count) + ": " + msg );
                              }
                    }
          }

          private static final String[] kPowerAlarms =
          {
                    "HUB COMMERCIAL POWER FAIL_FWD",
                    "HUB COMMERCIAL POWER FAIL",
                    "RBS COMMERCIAL POWER FAIL - Fixed_FWD",
                    "RBS COMMERCIAL POWER FAIL_FWD",
                    "RBS COMMERCIAL POWER FAIL - No Generator_FWD",
                    "RBS COMMERCIAL POWER FAIL - Portable_FWD",
                    "RBS COMMERCIAL POWER FAIL - Shared_FWD",
                    "RBS COMMERCIAL POWER FAIL - Yes_FWD",
                    "RBS COMMERCIAL POWER FAIL - YES_FWD",
                    "RBS COMMERCIAL POWER FAIL - Fixed",
                    "RBS COMMERCIAL POWER FAIL - No Generator",
                    "RBS COMMERCIAL POWER FAIL - Portable",
                    "RBS COMMERCIAL POWER FAIL - Shared",
```

```
                    "RBS COMMERCIAL POWER FAIL - YES",
                    "RBS COMMERCIAL POWER FAIL - Yes",
                    "RBS COMMERCIAL POWER FAIL",
                    "HUB COMMERCIAL POWER FAIL - Fixed",
                    "HUB COMMERCIAL POWER FAIL - No Generator",
                    "HUB COMMERCIAL POWER FAIL - Portable",
                    "HUB COMMERCIAL POWER FAIL - Shared",
                    "HUB COMMERCIAL POWER FAIL - Fixed_FWD",
                    "HUB COMMERCIAL POWER FAIL - No Generator_FWD",
                    "HUB COMMERCIAL POWER FAIL - Portable_FWD",
                    "HUB COMMERCIAL POWER FAIL - Shared_FWD",
          };
}
```

Filter Builder

MR server-side filtering allows a consumer to filter the stream of messages returned from the GET call. The following link provide details of building some of the filter to illustrate Filter Builder. It is not meant to cover and provide examples of every filter