This PDF is generated from authoritative online content, and is provided for convenience only. This PDF cannot be used for legal purposes. For authoritative understanding of what is and is not supported, always use the online content. To copy code samples, always use the online content.

Platform SDK Developer's Guide

Log Filtering

4/29/2025

Contents

Log Filtering

Introduction

Debug Log Level in Platform SDK protocol may affect Application performance due to huge log information output.

The aim is to introduce the ability to dynamically configure the verbosity of Platform SDK message logging. This way, production applications will be able to provide appropriate traces for troubleshooting without hurting performance with overly verbose logs.

This feature should provide ability to set message filtering, for defining which messages should be logged and which should not.

Message filter can be executed only when Debug log level is enabled.

Create and assign message filter directly

Use `setLogMessageFilter()` to assign custom log filter implementation for the protocol objects:

```
protocol.setLogMessageFilter(new MessageFilter(){
    public boolean isMessageAccepted(Message message) {
        if(message.messageId()==123) {
            return true;
        }
        else {
              return false;
        }
    }
});
```

This filter allows to log messages with ID equals to 123.

It is possible to assign default filter implementation, see described below.

Use Application Template to setup filters

Application Template provides default filter implementations. This filter can read configuration and handle updates from Configuration Server.

Default filter implementation should be wired with a client protocol using `FilterConfigurationHelper.bind()` method.

User needs to provide application name where filter configuration was defined and Config Service (to read application):

```
import com.genesyslab.platform.applicationblocks.com.ConfService;
import com.genesyslab.platform.applicationblocks.com.objects.CfgApplication;
import com.genesyslab.platform.applicationblocks.com.queries.CfgApplicationQuery;
import com.genesyslab.platform.apptemplate.configuration.ClientConfigurationHelper;
import com.genesyslab.platform.apptemplate.configuration.GCOMApplicationConfiguration;
import com.genesyslab.platform.apptemplate.configuration.IGApplicationConfiguration.IGAppConnConfiguration;
import com.genesyslab.platform.configuration.protocol.types.CfgAppType;
import com.genesyslab.platform.commons.protocol.Endpoint;
import com.genesyslab.platform.reporting.protocol.StatServerProtocol;

import com.genesyslab.platform.apptemplate.filtering.FilterConfigurationHelper;

public class MyApp {
   public void init() {

        ...
        //read application settings and create protocol
        String appName = "my-app-name";
        CfgApplication cfgApplication = confService.retrieveObject(
                   CfgApplication.class, new CfgApplicationQuery(appName));
        GCOMApplicationConfiguration appConfiguration =
                   new GCOMApplicationConfiguration(cfgApplication);
        IGAppConnConfiguration connConfig = appConfiguration.getAppServer(CfgAppType.CFGStatServer);

        Endpoint endpoint= ClientConfigurationHelper.createEndpoint(
                        appConfiguration, connConfig,
                        connConfig.getTargetServerConfiguration());

         StatServerProtocol statProtocol = new StatServerProtocol(endpoint);
         statProtocol.setClientName(clientName);

         //assign message filters to the protocol
         FilterConfigurationHelper.bind(statProtocol, appConfiguration, confService);

         statProtocol.open();
    }
}
```

> **Important**
>
> For manually created Endpoints the server host and port must match the server host and port of the `IGAppConnConfiguration` object (corresponds to one of the "Connections" tab entries in provided application). Otherwise the `ConfigurationException` "No connection object was found in application for protocol endpoint..." will be thrown. However this will not happen if the `Endpoint` has been created using the `ClientConfigurationHelper.createEndpoint()` helper.

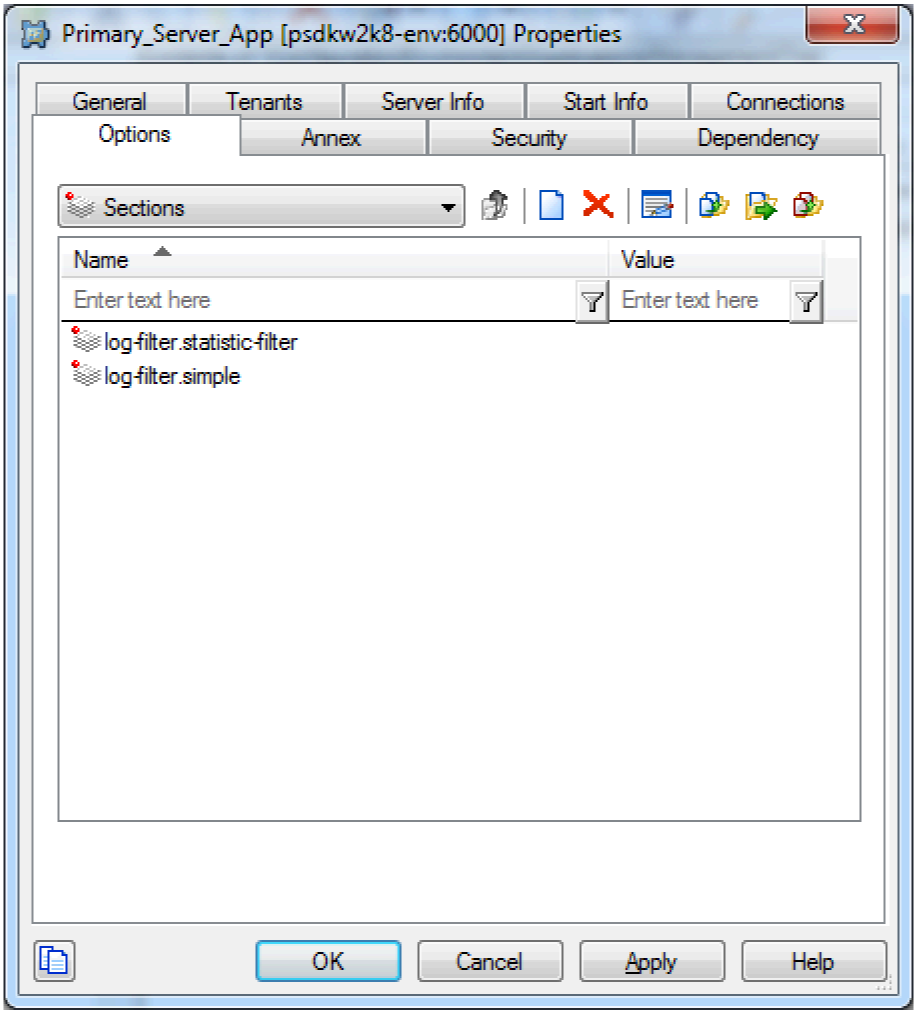
Helper method `FilterConfigurationHelper.bind()` reads application configuration, instantiates filter objects, assigns them to protocol, subscribe for Configuration Server notifications, registers handlers for protocol events and so on. When filters are not required anymore, release filtering infrastructure:

```
FilterConfigurationHelper.unbind(statProtocol,confService);
```

Use Configuration Manager to define filters, as described below.

Define filter in Configuration Manager

Filters are defined in the application "options" tab in configuration manager.



log-filter.statistic-filter
log-filter.simple

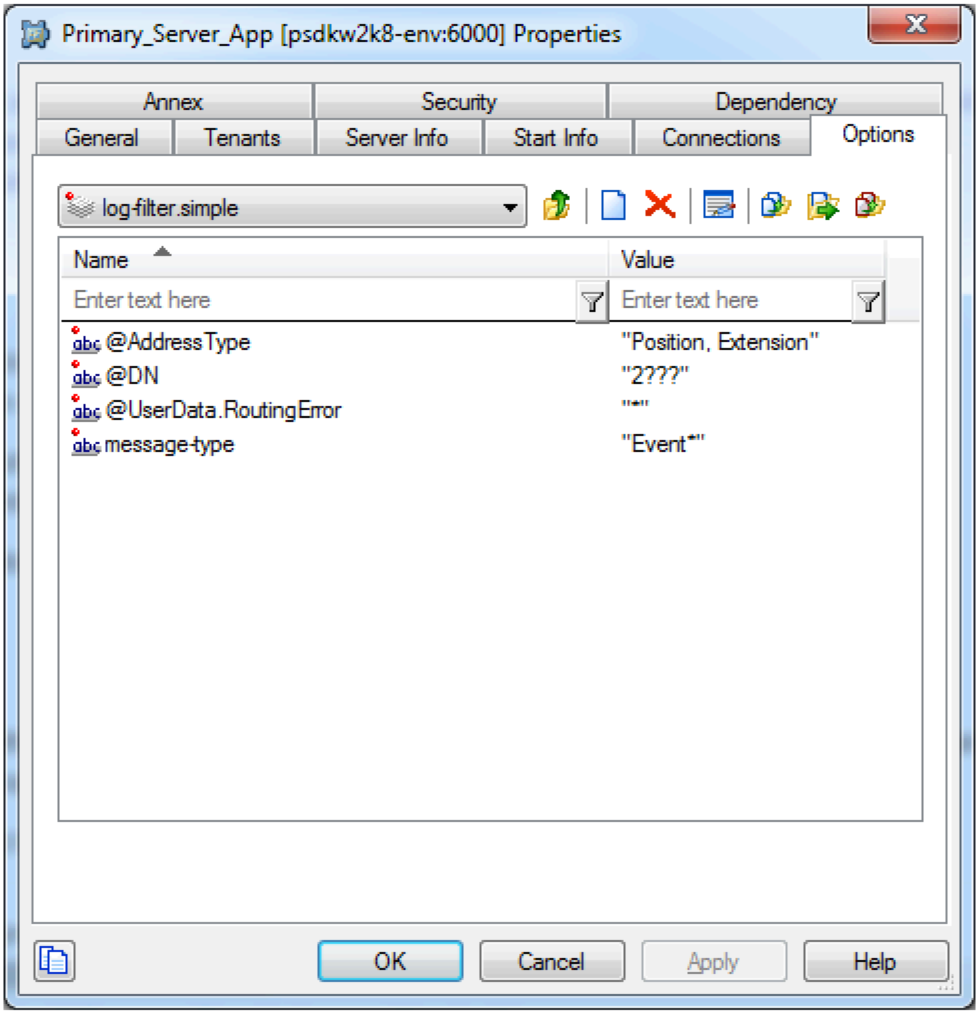
Filter name must be preceded with "log-filter." prefix.

For example: "`log-filter.my-filter`"

Filter names cannot start with "-", "!", or space symbols. Names such as "log-filter.-somefilter" are not allowed.

Filter options are specified under the "log-filter.name" section.

Example: Define filter for T-Server channel which will show only incoming events for DNs that are types of Positions or Extension and match "2???". Events should have user data with key "ROUTING_ERROR". Here is how configuration can be done:



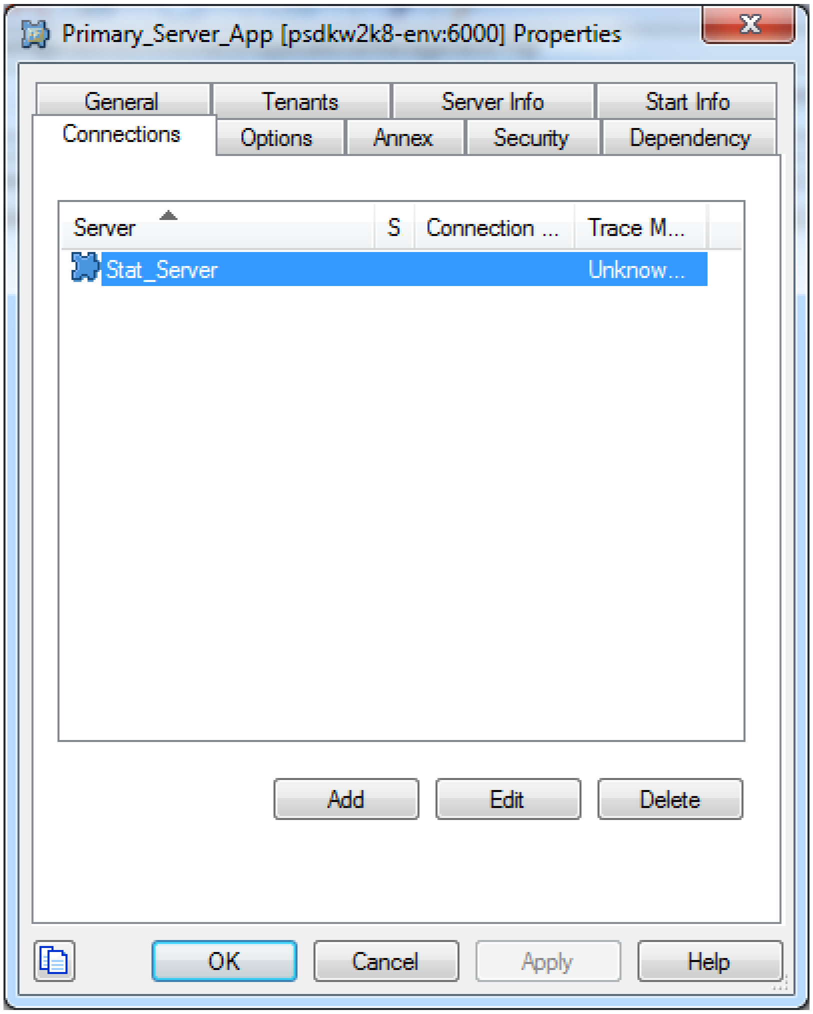
log-filter.simple

Name	Value
@AddressType	"Position, Extension"
@DN	"2???"
@UserData.RoutingError	"*"
message-type	"Event*"

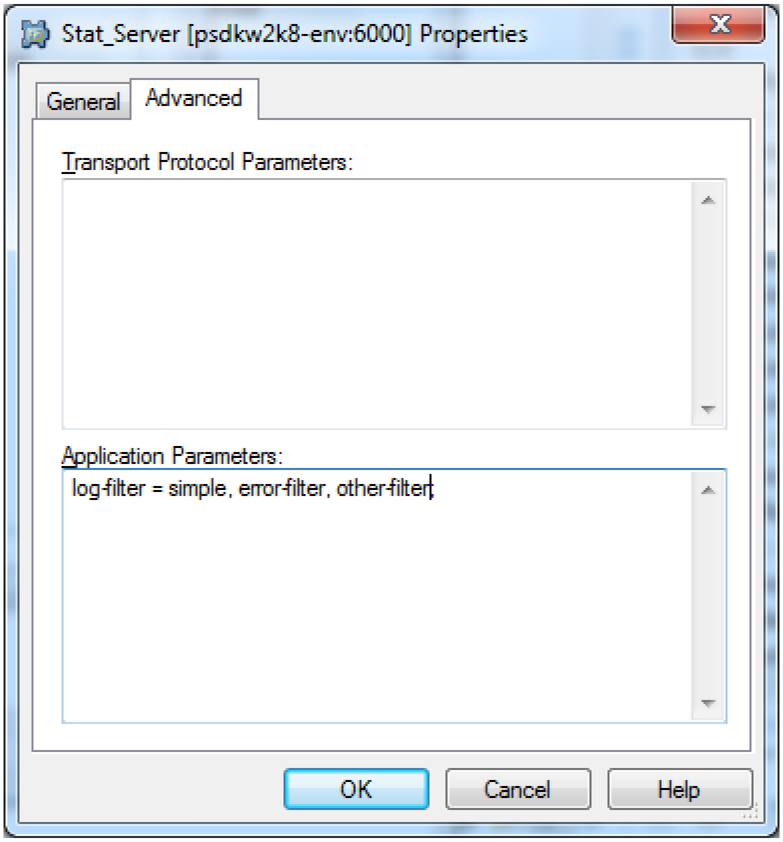
Filter options can represent one or more message attribute conditions.

```
log-filter.simple
    message-type = Event*
    @DN = 2???
    @AddressType = Position, Extension
    @UserData.RoutingError = *
```

After the filter is defined, assign it to one or more protocols on the application’s "Connection" tab.



Server	S	Connection ...	Trace M...
Stat_Server			Unknow...


Stat_Server [psdkw2k8-env:6000] Properties
General
Advanced
Transport Protocol Parameters:
Application Parameters:
OK
Cancel
Help

log-filter = simple, error-filter, other-filter

In Application Parameters it is possible to specify one or more filters for a protocol:

```
log-filter = simple, error-filter, other-filter
```

See Filter Chain for details.

Filter Syntax

It is possible to specify one or more conditions to filter messages by their names or (and) attribute values.

List of elementary conditions evaluate with "AND" operation.

Message Type Conditions

Evaluates message name or message id.

```
message-type = constant value
```

Example:

Option	Description
`message-type = EventInfo`	Filter accepting only EventInfo message
`message-type = 125,126,127`	Filter accepting messages with id 125,126 and 127.
`message-type = Event*`	Filter accepting all Events

Attribute conditions

Evaluates message attribute value.

```
@attribute-name = constant value
```

Example:

Option	Description
`@ReferenceId = 50`	Matches ReferenceId attribute value with 50.
`@DN = 2???`	Matches DN attribute value with 4-character string starting from "2". Attribute values like "2999" or "2ef7" will be accepted. Value "299999" will not be accepted.
`@UserData.CustomerID = 87624FAC`	Matches "CustomerID" key of the complex "UserData" attribute with "87624FAC" value.
`@StatisticObject.ObjectId = place*`	Matches statistic object which name start from "place"

Attribute names

Attribute name is specified after the "@" symbol. Attribute names should match getter name of the corresponding message class. To find out attribute name, see API reference guide for the corresponding message.

For example:

```
message.getStatisticObject().getObjectType().equals(...);
```

equals to filter condition

```
"@StatisticObject.ObjectType = ... "
```

To access sub element of the complex attribute (`KeyValueCollection, CompoundValue`), specify the name of inner element. Names should be delimited with "." symbol:

```
@attribute-name.element-1.element-n.
```

Supported attribute types

Currently supported message attribute types:

- string,
- int,

- enum,
- KeyValueCollection,
- CompoundValue (complex attributes like StatisticObject.)

If attribute has complex type, it is possible to specify matching condition only for one of its inner elements of a simple type: *string*, *integer* or *enum*. For example, here is how to specify condition for "TenantName" element of the complex "StatisticObject" attribute:

```
@StatisticObject.TenantName = 101
```

Constant values

Condition can have one or more constant values, delimited with ",". If one of the specified constants

matching attribute value (or message name), condition will return true.

Constant values supports wildcards:

Symbol	Description
*	Any sequence of symbols. For example, "Event*" matching any message name, starting from Event
?	Any symbol at specified position. For example, 555?5 will match 555A5 or 55505 strings.
\	Escape symbol. For example, 555\?5 will match only 555?5 string.

Constant examples:

```
@AgentPlace =  place1000
```

```
@AgentPlace = place*
```

```
@AgentPlace = place100??
```

```
@AgentPlace = place110, place120, place2??
```

Inverted conditions

It is possible to invert attribute condition or message condition result by specifying "#not" prefix:

```
#not message-type = RequestAgentLogin
```

or

```
#not @DN = 10
```

Empty filters

Filter with empty options will return negative evaluation result. It can be used to deny all messages for a protocol object.

Stateful filters

Sometimes user doesn't know what attribute value should be specified in a filter condition. For example, in Statistic protocol, the `ReferenceId` attribute (which uniquely identifies `EventInfo` message with statistic data), initializing at a runtime, during statistic opening. To find out `ReferenceId` value, user needs to search corresponding `RequestOpenStatistic` in logs.

Use case: trace `EventInfo` messages with any statistic for "place100".

Default log filter implementation allows to save attribute value from one message and re-use it to trace other messages.

Configuration sample:

```
log-filter.stat-filter
    message-type = RequestOpenStatistic
    @StatisticObject.ObjectId = place100
    trace-on-attribute = ReferenceId
    trace-until-message-type = EventStatisticClosed
```

When filter meets condition ("RequestOpenStatistic" message with "place100" statistic object), the ReferenceId is added to the list of saved values.

When filter receives message ("EventInfo" in statistics protocol) with `ReferenceId` matching to any of the saved values, it allows to log the messages. More then one statistics for "place100" could be opened, so it is possible to store several `ReferenceId` values.

When filter receives "EventStatisticClosed" with `ReferenceId` matching to any of the previously saved values, this value is removed from the list. When values list is empty, no messages could be logged.

Note 1: Filters processing messages only when debug log level is enabled. In order to save `ReferenceId` value, debug log level should be enabled before `RequestOpenStatistic` is sent to server.

Note 2: Saved values are cleared upon protocol close.

Note 3: Number of saved values is limited to 1024 to prevent high memory consumption upon incorrect filter conditions. It can be changed with system property "com.genesyslab.platform.filtering.valuelist.capacity". Changing to greater value is not recommended.

Filter Chain

List of filters on the "Connection tab" represent a filter chain. By default, filter chain evaluates filter results as "OR" expression. If one of the filter accept message message will be logged and other filters will not be evaluated. Filters can be of two types: "accept" and "deny" filters

Use cases

Filter type	Use case
accept (default)	User exactly knows criteria by which messages should be logged. For example, log all Events and Requests with "DN" attribute "2000".
deny	User see a lot of unneeded messages in log with common data. User can specify "deny" filter to truncate those messages.

Filter chain behavior

Filter type	Message evaluation result	Filter chain behavior
accept (default)	TRUE\FALSE	If TRUE - allow log, do not execute other filters. Otherwise, execute next filter. Deny log if last filter returned FALSE
deny	TRUE\FALSE	If TRUE - deny log, do not execute other filters. Otherwise, execute next filter. Allow log if last filter returned FALSE

Syntax

Filter name, prefixed with "-" means "deny" filter. Names without prefix mean "accept" filter.

Example:

```
"log-filter = -filter"
```

```
"log-filter = f1, f2, -f3".
```

Filter result negation

Optionally, it is possible to invert message evaluation result for a filter with "!" symbol.

Example:

```
"log-filter = !filter"
```

```
"log-filter = f1, !f2, f3"
```

```
"log-filter = f1, -f2, -!f3".
```

Special filters

While delivering message from TCP connection to the client's receiver (or in opposite direction), Platform SDK can trace message on the different points of its way:

```
2014-07-31 15:07:38,168 [New I/O worker #1] DEBUG otocolMessagePackagerImpl  - New message #2
....
2014-07-31 15:07:38,168 [New I/O worker #1] DEBUG ns.protocol.DuplexChannel  - Complete
message handling: 2
```

It is possible to disable such log entries with special filter "skip-trace-msg". This filter can be specified as a stand-alone filter, or can be used together with other filters in a filter chain:

Example:

```
log-filter = skip-trace-msg
log-filter = filter-1, filter-2, filter-n, skip-trace-msg
```