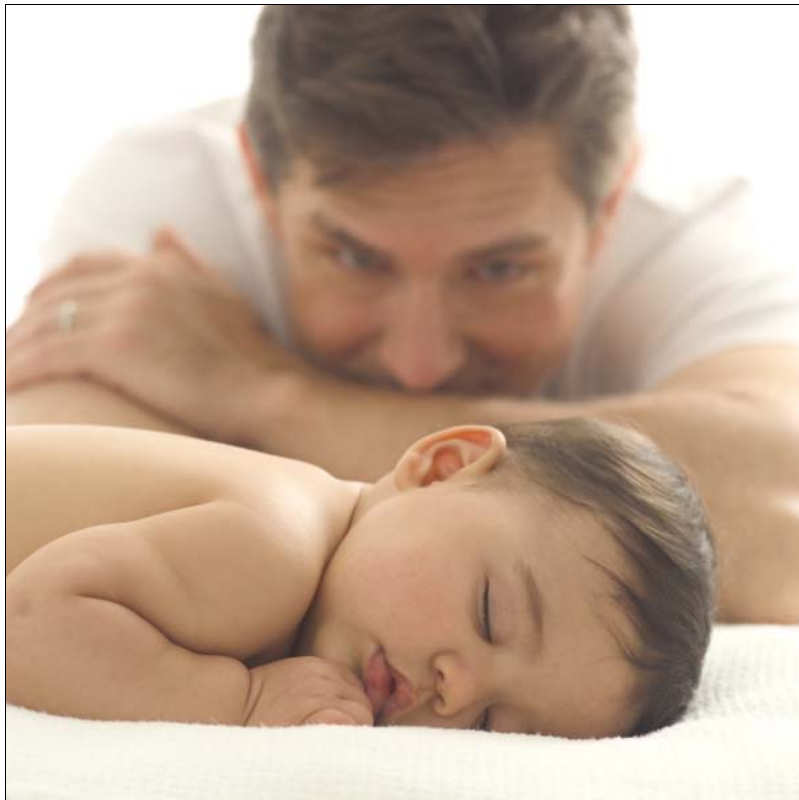


ESA Performance Management

Ericsson SNMP Agent 16.0

SYSTEM ADMINISTRATION GUIDE



Copyright

© Ericsson AB 2004-2016. All Rights Reserved.

Disclaimer

No part of this document may be reproduced in any form without the written permission of the copyright owner.

The contents of this document are subject to revision without notice due to continued progress in methodology, design and manufacturing.

Ericsson shall have no liability for any error or damage of any kind resulting from the use of this document.

Trademark List

Ericsson is the trademark or registered trademark of Ericsson AB. All other products or service names mentioned in this document are trademarks of their respective companies.



Contents

1	About This Document	1
1.1	Purpose	1
1.2	Target Group	1
1.3	Prerequisites	1
1.4	Typographic Conventions	1
2	Introduction	3
2.1	Overview	3
2.2	Configuration Work Process	4
2.3	Groups and Counters	5
2.4	Data Sources	6
2.5	Output	7
2.6	Thresholds	7
3	Define a Counter	9
3.1	Introduction	9
3.2	Function Description	9
3.3	Configuration Format	12
3.4	Configuration Deployment	26
3.5	Configuration Activation	27
3.6	Examples	27
4	Define a Job	29
4.1	Introduction	29
4.2	Function Description	29
4.3	Configuration Format	29
4.4	Configuration Deployment	34
4.5	Configuration Activation	34
4.6	Examples	34
5	Define a Threshold	35
5.1	Overview	35
5.2	Function Description	35
5.3	Configuration Format	38
5.4	Configuration Deployment	44



5.5	Configuration Activation	44
5.6	Examples	44
6	Output SNMP	45
6.1	Introduction	45
6.2	SNMP MIB Format and Content	45
7	Output 3GPP XML File	47
7.1	Introduction	47
7.2	File Format and Content	47
8	Administration	49
8.1	Log Files	49
8.2	Backup and Restore	49
9	Command Line Interface	51
9.1	Introduction	51
9.2	Command: Counter Status	51
9.3	Command: Reset Counter	52
9.4	Command: Verify Data Source	53
9.5	Command: Read Counter	54
9.6	Command: Write Counter	56
9.7	Command: View Job	57
9.8	Command: View Threshold Alarms	60
10	Application Programming Interface	63
10.1	Introduction	63
10.2	Interface: Counter	63
10.3	Interface: Job	63
10.4	Interface: Threshold	63
10.5	Interface: Output Reporter	64
	Glossary	67
	Reference List	69



1 About This Document

1.1 Purpose

The purpose of this document is to describe the system administration tasks for the Performance Management Agent (PMA) within the product Ericsson SNMP Agent (ESA). The document contains description about how to configure the PMA for gathering and publishing PM counters.

1.2 Target Group

The target group for this document is personnel responsible for administration of the ESA.

1.3 Prerequisites

It is assumed that the user of this document fulfils the following prerequisites.

- Is familiar with XML.
- Has system administrator authority to the server, in which the ESA is installed.

1.4 Typographic Conventions

The typographic conventions used in this document are described in Reference [1].





2 Introduction

2.1 Overview

The configuration of the PM Agent (PMA) in the ESA is about setting up one or several of the following configurations.

☐ **Counter Definition**

This is about defining all the counters to collect and made available by using the ESA PMA. Each counter is given a unique identity, a name and description, and, if available, the collection operation to perform to read data from the data source.

☐ **Job Definition** *(optional)*

This is about defining the jobs, which are creating PM data files containing the collected counter data. Each job contains information about the counter group to publish and how often to publish it. If file output is not wanted, the job definition is not needed.

☐ **Threshold Definition** *(optional)*

This is about defining counter thresholds, which will trigger alarms if thresholds are breached. When a counter is too high or low breaching a defined threshold, a SNMP alarm can be sent. If threshold alarms are not wanted, the threshold definitions are not needed.

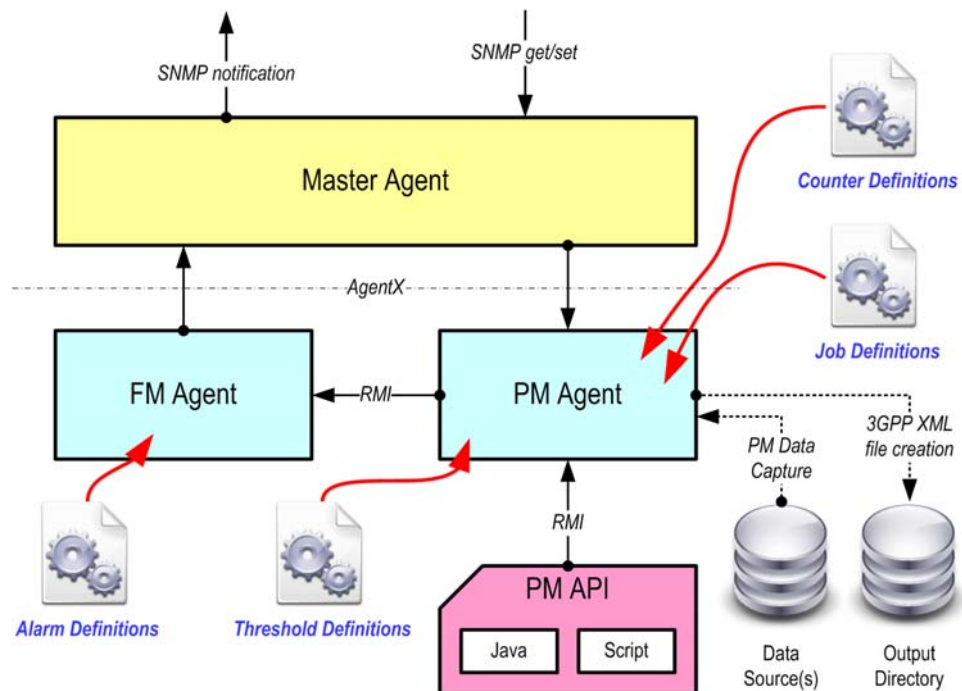


Figure 1 An overview of the configuration needed to get the ESA PMA in operation.

The figure above gives an overview of the three configuration files needed for setting up the PMA.

At least one *counter definition* must be created to get the PMA into operation. It makes the PMA operational and also the counter is made visible on the SNMP interface, which means an OSS can read PM data using a SNMP get operation.

If a 3GPP XML file is wanted as output, a *job definition* must be created.

Finally, to get alarms when a threshold is reached on a counter, a *threshold definition* must be created. Please note that creating a threshold definition means that also the alarm to trigger must be defined. This is done as an *alarm definition* in the FM Agent, see Reference [3].

All configuration files are created using XML syntax.

2.2 Configuration Work Process

The ESA PMA is setup by executing the following work process:

1. Define PM counter(s).

Described in Section 3 on page 9.

2. If PM data output to file is wanted, defined job(s).



Described in Section 4 on page 29.

3. If PM threshold alarms are wanted, define threshold(s).

Described in Section 5 on page 35.

2.3 Groups and Counters

A “Group” is a collection of counters. The group concept is used to give the possibility to group counters that are similar to each other or that in some way belong or have a relation to each other. From ESA point of view there is no restriction on how to use groups. To get the PMA into operation at least one group must be defined and within the group there must be at least one counter.

A “Counter” is a single value supporting only non negative numbers up to value $2^{63}-1$. Each counter defined in the PMA must be connected to a group. The ESA as such does not care, does not know and does not understand what the counter is used for. The ESA is simply responsible for collecting the counter value and making it accessible for one or several OSS systems.

If the PMA gets a non-valid value, such as a string or a null value, the operation is considered erroneous and logged. The PMA does however not update the counter value in the PMA storage (as no new valid value was given) and continues to work as normal. If a data source is defined, the ESA will at each interval try to read PM data from the data source, no matter if the data collection operations are erroneous and/or give bad values.

A counter is either of the following two types:

- **Cumulative Counter**

The counter maintains a running count of the event being counted, which means this counter is only increased and never decreased. The counter rotates when the counter reaches the maximum value of its data type, such as max value is 65535 for a counter with data type `integer16`, which means it starts over at zero.

When looking at counter sample values within a period this counter type has a delta value for the period. The delta value is part of the output created from defined PM jobs.

The delta value is always a positive number and if the actual value is lower than the value in the previous measurement period, the delta will be same as actual value.

Counter example: A counter representing "number of messages sent" is always increasing. That is a *cumulative counter*.

- **Gauge Counter**

Gauges represent dynamic counters that may change in either direction, which means they are both increased and decreased.



When looking at counter sample values within a period this counter type has maximum and a minimum values for the period. The maximum and minimum values are part of the output created from defined PM jobs.

Example: A counter representing "the CPU load" may be both increased and decreased. That is a *gauge*.

2.4 Data Sources

2.4.1 Description and Definition

Each counter may have a data source from which the PMA is reading the counter value. The data source can be one of the following:

CSV	The ESA may capture the value from any column from the last line in any CSV file.
JMX	The ESA may capture the value from any JMX MBean in the system.
Script	<p>The ESA may capture the value returned from any script.</p> <p>Please note that this implies that an <i>executable</i> of some kind must be created that returns the data of interest and the returned data must be a single value only.</p>
SNMP	The ESA may capture the value from any SNMP leaf from any SNMP agent on any IP address.
XML	The ESA may capture a value from any XML element or attribute in any XML file using Xpath expressions.

The ESA has the possibility to support a multi counter script, which means a shell script can be executed to return more than one value in one operation. This is useful for large data sources. Instead of setting up the ESA to read the counters one by one, all the counters could be fetched at once.

2.4.2 Data Capturing Interval

For each data source defined there is a capture interval defined. The interval specifies at which interval the ESA shall read data from the data source. The interval is defined in seconds and the following values are allowed; 10 | 15 | 20 | 30 | 60 | 120 | 300 | 600 | 900 | 1200 | 1800 | 3600. Keep in mind that when using the lower values the ESA (and the system) is put into a more heavy load as the data collections occur more frequently.

The ESA is synchronizing the data capturing within an hour. That means that for example a 5 minute definition (300 seconds) captures the data at even 5



minute periods within an hour. Even though the PMA for example is started at 12:03:41 the PMA starts the capturing of data from the data sources at 12:05, 12:10, 12:15 and so on. The reasoning is also valid for intervals in seconds within a minute.

Note: Please note that if the system date or time is change, the ESA PMA needs to be restarted in order to prevent 3GPP reports with faulty date and time.

2.5 Output

The ESA supports the following outputs of the managed counters:

SNMP	The counters are reported on the SNMP interface. See Section 6 on page 45.
XML	The counters are reported to file in XML format following 3GPP standards. See Section 7 on page 47.
Custom	The counters may be reported in any format, such as to an own file format or to a SQL database. This is possible by using the PM API for creating a customized output reporter. See Section 10.5 on page 64.

The SNMP interface is useful for OSS systems that needs PM data more close to real time. As the SNMP is a direct read operation the value is instantly collected from the ESA and can immediately be used by the OSS. However, keep in mind that the SNMP interface only holds the last value. Historical representation of PM data on SNMP is not available.

The XML file based format is useful for reporting tools, which are harvesting data from systems in a network and then processes the data for later use, such as for creation of reports. The files contain a history of data.

2.6 Thresholds

The ESA supports definition of thresholds on the PM data. When a counter rises above the threshold the ESA may send an *alarm raise*. And, when a counter falls below the threshold the ESA may send an *alarm clear*. By using thresholds in the PMA it means Fault Management operations are introduced in your system by setting up the PMA.

If the correlation of alarm raise and alarm clear is not wanted, an *event* can be sent instead. The event is in such case sent both when the threshold is



breached and when the value falls back below the threshold. The event content indicates what has happened.



3 Define a Counter

3.1 Introduction

The counter definition is about creating unique counters that are used to represent levels, statuses and statistics in the system being monitored. The ESA is managing the counters by holding the PM data, updating the PM data and reporting the PM data.

3.2 Function Description

When setting up the PMA, the data sources holding the PM data of interest must be known. Then decisions must be taken about which counters are to be managed using the PMA. When the PM counters are known and the data sources are known, it is time for setting up the Counter Definition(s) in the PMA.

Each “counter” always belong to a “group”. The group is used for two purposes; grouping similar objects together and for identifying output of data to file when setting up jobs. The “group identity” and “counter identity” are concatenated, such as `GROUP: COUNTER`, always a unique identity of each counter. They may never occur twice in the PM configuration. When a counter is created it is either *concrete* or *abstract*. A concrete counter is a single counter holding a single value. An abstract counter does not hold any values, but can be instantiated where each *counter instance* is holding a value.



A counter group is created to hold a set of counters related to hardware monitoring. The following group name is defined.

HARDWARE

The group of counters shall hold the values for CPU load, Memory usage and Disk usage. The following counters are defined.

CPULOAD
MEMUSAGE
DISKUSAGE

The PMA will now hold three unique counters identified with the following concatenated identities.

HARDWARE:CPULOAD
HARDWARE:MEMUSAGE
HARDWARE:DISKUSAGE

The three counters are now defined and unique in the PMA.

Example 1 A counter grouping and identification example using concrete counters.



A counter group is created to hold three abstract counters handling call counts for clients managed in the system. The following group name is defined.

```
CLIENT
```

The counter shall hold the value for number of call attempts, number of successful calls and number of unsuccessful calls. The following counters are defined.

```
CALLATTEMPTS
CALLSUCC
CALLUNSUCC
```

The PMA will now manage three unique abstract counters identified with the following concatenated identities.

```
CLIENT:CALLATTEMPTS
CLIENT:CALLSUCC
CLIENT:CALLUNSUCC
```

When a new client CLIA is introduced in the system, the PMA will manage three new counters where all are instantiated from the abstract counter.

```
CLIENT:CALLATTEMPTS:CLIA
CLIENT:CALLSUCC:CLIA
CLIENT:CALLUNSUCC:CLIA
```

With two additional clients CLIB and CLIC there will be the following counters.

```
CLIENT:CALLATTEMPTS:CLIB
CLIENT:CALLSUCC:CLIB
CLIENT:CALLUNSUCC:CLIB
CLIENT:CALLATTEMPTS:CLIC
CLIENT:CALLSUCC:CLIC
CLIENT:CALLUNSUCC:CLIC
```

With only three defined abstract counters the PMA now handles nine counter values in total.

Example 2 A counter grouping and identification example using abstract counters.

A counter in the PMA may get the value from two different interfaces:

- The Pull mechanism: The PMA *pulls* the data from a data source.
- The Push mechanism: The PMA API is used by an application, which *pushes* counter values to the PMA.

If the PMA is setup to fetch the data, the PMA is configured to do so with an interval. When the PMA is capturing data from a data source and it fails, the error is logged and the PMA continues to work as normal.

The PMA API is described in Section 10 on page 63.

The collection of data from data sources work in two modes:

- **Single counter capture**

One counter is defined with a single data source, which means the data source returns a single value. The counter can also be defined with no data source at all, which means the PM API is used to push values into the PMA for that counter.

- **Multiple counter capture**

Several counters are defined with a single data source, which means the data source returns multiple values at a single capture operation. This is named a *counter collection* in the PMA. The API can still be used for counters in a collection, if needed.

The multiple counter option should be used if a large number of counters are defined. The load on the ESA is kept lower by providing several counters at one capture operation than using one capture operation for each counter. This is also known as a “Collection of counters”, not to be mistaken with the definition of “Groups”.

3.3 Configuration Format

3.3.1 Overview

The figure below shows a high-level overview of the counter definition configuration file.

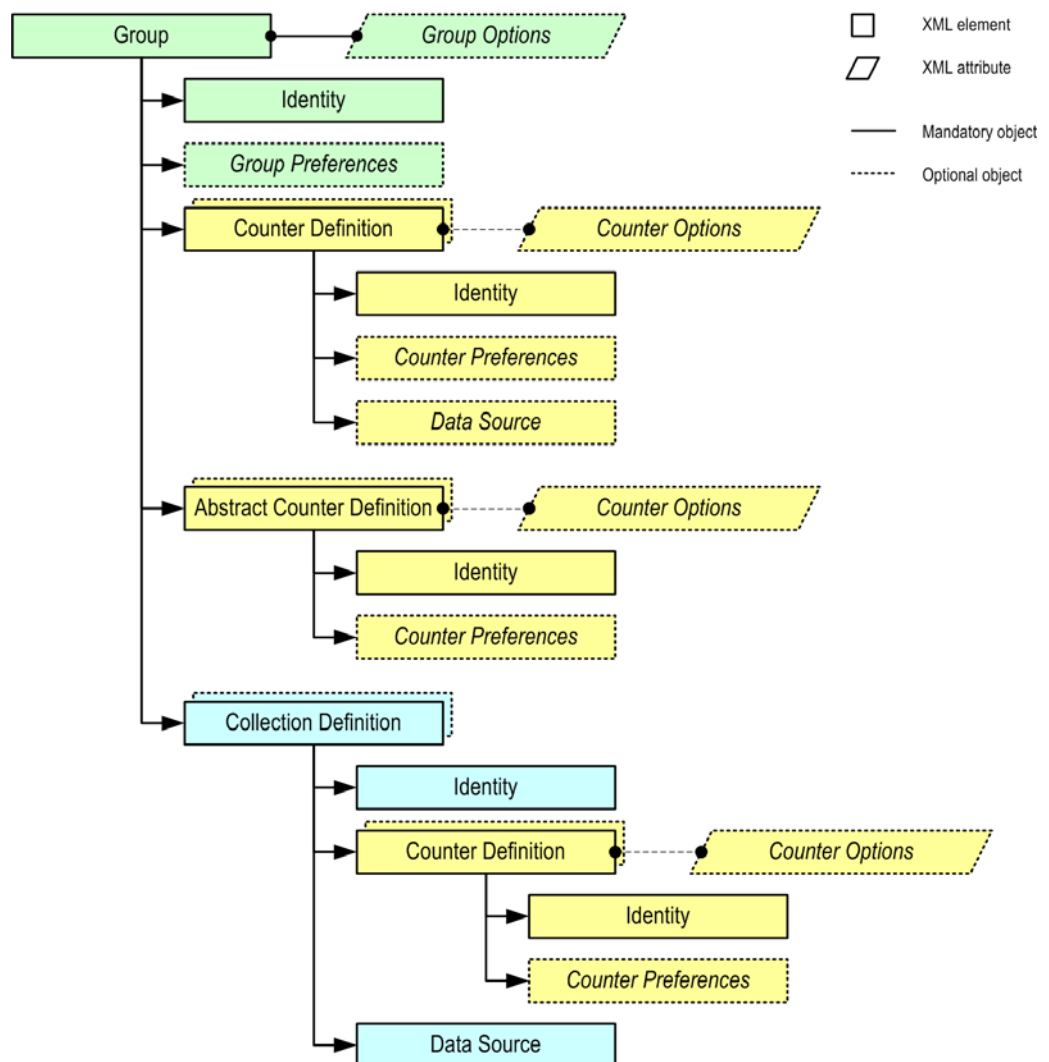


Figure 2 An overview of the XML objects used to create one or several PM counters in the ESA PMA.

One *group* contains one or several *counters* and/or one or several *counter collections*. The group is described by giving it a unique identity and a description. The counter is defined by a unique identity within the group and a description. If a data source in the system is holding the counter value, the data source is defined. If there is a script developed to return several counter values in one execution a counter collection can be defined. The collection is defined with a number of counters and the one and only data source. The details of the counter definition configuration file are found in the following chapters.

3.3.2 Group Definition

The following is an example of how to define a PM counter group:

```
<pmCntGroup active="yes">
```

```

<identification>
  <groupId>MXDB</groupId>
</identification>
<description>
  <groupDescr>Message Transfer Database</groupDescr>
  <groupInfo>The counters show the database usage for
              message handling processes.</groupInfo>
</description>

```

...more...

</pmCntGroup>

- pmCntGroup

This is the element holding the configuration of a group of counters.

- active

This attribute is for activating or deactivating the group. Instead of removing the configuration of a group when the group is of no interest, this attribute is simply set to “no” to make it inactive. The result is that the data is not collected, the counters in the group can not be viewed and jobs can not be defined for the group.

yes	The group is active.
no	The group is inactive.

- identification

This is the group identification that makes the group unique.

- groupId

This is the identity of the group. Use a short and useful name that uniquely identifies the group.

Valid characters are A to Z, a to z and 0 to 9 using no spaces. The length is restricted to 2-32 characters.

The characters "-", "_", "/", "\", "@", and "." are in fact also allowed, but not recommended. Using these characters may result in usability issues, such as CLI usage and CLI printouts. Please keep in mind that the name should be chosen *to simplify for the technicians that are managing the counters to understand the counter purpose and content*. The name should not be chosen only to simplify design and implementation.

Examples:

groupId = HARDWARE



```
groupId = DATABASE
groupId = CALLS
```

- description (optional)

These are place holders for a more detailed textual information about the group.

– groupDescr

A slogan or title for the group.

Examples:

```
groupId      = HARDWARE
description   = System Hardware Statistics
```

```
groupId      = DATABASE
description   = MySQL Database Performance
```

```
groupId      = CALLS
description   = Client Call Statistics
```

– groupInfo

Additional group information. For example; What does the group represent? What kind of data does it contain?

Examples:

```
groupId      = HARDWARE
information    = This group holds counters related to
                  the system hardware. These should be monitored
                  to capture hardware faults early in order to
                  secure a stable system.
```

```
groupId      = DATABASE
information    = This group holds database performance
                  counters. These should be used to keep the
                  database healthy and secure that data processing
                  is fast.
```

```
groupId      = CALLS
information    = This group holds call related counters.
                  Too many lost and dropped calls may indicate
                  network problems that should be taken care of in
                  order to secure call revenues.
```

3.3.3 Single Counter

The following is an example of a defined counter within a defined group:

```
<cntDefinition active="yes" activeSnmpp="yes" cntType="CC">
  <identification>
    <counterId>MYCOUNTERCSV</counterId>
  </identification>
  <description>
    <counterDescr>A description of the counter</counterDescr>
    <counterInfo>Additional counter information</counterInfo>
  </description>
  <preferences>

    ...counter value preferences...

  </preferences>
  <dataSource interval="60">

    ...data source definition...

  </dataSource>
</cntDefinition>
```

The XML element and attribute definitions:

- cntDefinition

This element holds the configuration of a counter.

- active

This attribute is for activating and deactivating the counter. Instead of removing the configuration of a counter when the counter is of no interest, this attribute is simply set to “no” to make it inactive. The result is that the data is not collected, the counter can not be viewed and jobs can not be defined for the counter.

yes	The counter is active.
no	The counter is inactive.

- activeSnmpp

This attribute is for activating and deactivating the presentation of the counter on the SNMP interface. If it is set to “no” it is simply not possible to view the counter on the SNMP interface.

yes	The SNMP view is active.
no	The SNMP view is inactive.

- cntType

This attribute specifies the counter type.



CC Cumulative	The counter type is 'Cumulative Counter'.
GC Gauge	The counter type is 'Gauge Counter'.

- `identification`

This is the counter identification that makes the counter unique.

- `counterId`

This is the counter identity. Use a short and useful name that uniquely identifies the counter within the defined group.

If there are more counters defined within the same group having the same name, the PMA will discard the duplicate counter.

Valid characters are A to Z, a to z and 0 to 9 using no spaces. The length is restricted to 2-64 characters.

The characters "-", "_", "/", "\", "@" and "." are in fact also allowed, but not recommended. Using these characters may result in usability issues, such as CLI usage and CLI printouts. Please keep in mind that the name should be chosen *to simplify for the technicians that are managing the counters to understand the counter purpose and content*. The name should not be chosen only to simplify design and implementation.

Examples:

```

groupId = HARDWARE
    counterId = CPULOAD
    counterId = DISKUSAGE
    counterId = MEMUSAGE

groupId = DATABASE
    counterId = DBSESSIONS
    counterId = TABLESPACEUSAGE

groupId = CALLS
    counterId = CALLATTEMPTS
    counterId = CALLSUCC
    counterId = CALLUNSUCC
    counterId = LOSTCALLS

```

- `description (optional)`

These are place holders for a more detailed textual information about the counter.

- `counterDescr`

A slogan or title for the counter.

**Examples:**

```
groupId = HARDWARE
  counterId      = CPULOAD
  description    = The CPU Load (%)

  counterId      = DISKUSAGE
  description    = The disk usage (%)

  counterId      = MEMUSAGE
  description    = The memory usage (MB)
```

– counterInfo

Additional counter information. For example; What does the counter represent? What kind of data does it contain? What is the behavior of the counter?

Examples:

```
groupId = HARDWARE
  counterId      = CPULOAD
  information    = This counter indicates the
                  processing load of the system. In case
                  the load is above 95% for a long period
                  of time it is recommended to enhance the
                  processing capacity of the system.

  counterId      = DISKUSAGE
  information    = This counter indicates the
                  disk usage of the system. In case the
                  disk usage is above 80% it is recommended
                  to enhance the disk capacity of the
                  system.

  counterId      = MEMUSAGE
  information    = This counter indicates the
                  memory usage of the system. In case the
                  memory usage is above 30MB for a longer
                  period of time it is recommended to kill
                  hanging processes in order to free up
                  memory.
```

- preferences (optional)

This is the counter value preferences, such as level and limit values, valid for the counter.

Most of the data defined using the following XML elements are used for documentation only and do not affect the behavior or output of the PM Agent. The benefit of using the following elements is to make it easier for the reader of the configuration file to understand the counter and the



configuration settings. If the data defined is actively used in the PMA it is clearly stated.

- `unit` (optional)

The unit of the PM counter value, for example “%” or “transactions per sec”.

- `resetValue` (optional)

The value of the PM counter when being reset. The reset operation is performed using the PM API.

- `minValue` (optional)

The minimum value of the PM counter.

- `maxValue` (optional)

The maximum value of the PM counter.

- `dataSource` (optional)

This is the definition of the data source for the counter.

- `interval`

This attribute defines the interval in seconds between the capturing of PM data operations from the data source.

Possible values: 10, 15, 20, 30, 60, 120, 300, 600, 900, 1200, 1800 and 3600.

Please use the lower values with care. Capturing PM data with a too short interval may overload the ESA as well as the system, which may lead to that the capture time becomes longer than the interval time. Such situation may lead to unexpected PMA behavior and corrupt PM data.

- `csv`

The ESA collects the value found in the column which is separated with the defined field separator from the last line in the defined file.

`location` The path to the CSV file.

`fieldSeparator` The field separator used in the CSV file.

`columnIndex` The column in the CSV file where the PM value is read.

Example:

<CSV>



```
<location>/<path>/file.csv</location>
<fieldSeparator>;</fieldSeparator>
<columnIndex>3</columnIndex>
</csv>
```

– jmx

The ESA collects the value found in the defined JMX bean.

url	The JMX bean URL.
objectName	The JMX bean object name.
attribute	The JMX bean attribute.
userName	The JMX bean username.
password	The JMX bean password.

Example:

```
<jmx>
  <url>
    service:jmx:rmi:///jndi/rmi://localhost:9999/server
  </url>
  <objectName>
    DefaultDomain:type=com.ericsson.system.pmcounters,index=1
  </objectName>
  <attribute>Transactions</attribute>
  <userName>myuser</userName>
  <password>mypw</password>
</jmx>
```

– script

The ESA collects the value returned from the execution of the specified command line. The use of `script` means the ESA is executing the command line given to the parameter `location`.

It is highly recommended to create the data collection shell script with some kind of execution control. If the script takes too long time to execute, the PMA will take action. If the script is known to have a long execute time, the solution is to either rewrite the script or select a longer data collection interval.

The PMA has been built with execution and time control. If the script execution time *is longer than half the defined interval*, the ESA will kill the script. At every interval the PMA will continuously trigger the script even though it may be killed due to long execution time. If the script is killed, the counter value is not updated for that interval.

When triggering the script, please be aware of that all environment variables but `TERM`, `PATH` and `LANG` are discarded. This is done by service handling of the PMA service and not by the ESA itself. If you



need to use environment variables within the script, they must be sourced within the script.

The ESA collects the value returned by the defined shell script.

location The command line to execute, which returns the counter value. Arguments may be used in the command line.

Example 1: Shell script

```
<script>
  <location>/<path>/myScript.sh</location>
</script>
```

Example 2: Java

```
<script>
  <location>/<path>/java -jar getMyCounter.java</location>
</script>
```

Example 3: Perl script

```
<script>
  <location>/<path>/myPerlScript.pl</location>
</script>
```

— snmp

The ESA collects the value found in the defined SNMP object identity from the SNMP agent defined using IP address, port number and community string. The SNMP get operation supports SNMP v2c.

agentIP The IP address of the SNMP agent.
agentPort The port number of the SNMP agent.
agentComStr The community string of the SNMP agent.
oid The SNMP object identity identifying the PM data.

Example:

```
<snmp>
  <agentIP>10.10.1.99</agentIP>
  <agentPort>20161</agentPort>
  <agentComStr>public</agentComStr>
  <oid>.1.3.6.1.4.1.1977.9.1.29</oid>
</snmp>
```

— xml

The ESA collects the value returned by running a defined Xpath expression on a specified XML file.



file The path to the XML file holding the PM data.
xpath The Xpath expression capturing the PM value.

Example:

```
<xml>  
  <file>/<path>/file.xml</file>  
  <xpath>/transactions/sms/chars[last()]</xpath>  
</xml>
```

3.3.4 Multiple Counter

The following is an example of a defined collection holding several counters as part of a defined group:

```
<cntCollection active="yes" activeSnmp="yes">  
  <identification>  
    <collectionId>MSGPROCCOLLECTION</collectionId>  
  </identification>  
  <cntDefinition cntType="CC">  
    <identification>  
      <counterId>MSGPROC1</counterId>  
    </identification>  
    <description>  
      <counterDescr>Message process 1</counterDescr>  
      <counterInfo>The number of messages processed by process  
                    engine 1 in system XYZ.</counterInfo>  
    </description>  
    <preferences>  
      ...counter preferences added here...  
    </preferences>  
  </cntDefinition>  
  <cntDefinition cntType="CC">  
    <identification>  
      <counterId>MSGPROC2</counterId>  
    </identification>  
    <description>  
      ...counter description added here...  
    </description>  
    <preferences>  
      ...counter preferences added here...  
    </preferences>  
  </cntDefinition>  
  <cntDefinition cntType="GC">  
    <identification>  
      <counterId>MSGPROC3</counterId>  
    </identification>  
    <description>  
      ...counter description added here...  
    </description>
```



```

    <preferences>
      ...counter preferences added here...
    </preferences>
  </cntDefinition>
  <dataSource interval="60">
    <script>
      <location>/opt/path/script</location>
    </script>
  </dataSource>
</cntCollection>

```

The example above defines three counters and the counter values are received by the one and only script defined, which means one script returns three values.

The XML element and attribute definitions:

- `cntCollection`

This is the element holding the configuration of a collection of counters having the same script as data source.

- `active`

This attribute is for activating or deactivating the collection. Instead of removing the configuration of a collection when the collection is of no interest, this attribute is simply set to “no” to make it inactive. The result is that the data is not collected, the counters in the group can not be viewed and jobs can not be defined on the group.

yes	The collection is active.
no	The collection is inactive.

- `activeSnmpp`

This attribute is for activating and deactivating the presentation of the counters that belong to the collection on the SNMP interface. If it is set to “no” it is simply not possible to view the counters using SNMP get operations.

yes	The SNMP view is active.
no	The SNMP view is inactive.

- `identification`

This is the collection identification that makes the collection unique.

- `collectionId`

This is the collection identity. Use a short and useful name that uniquely identifies the collection within the defined group. Two or more collections with the same identity are not allowed.

Valid characters are A to Z, a to z and 0 to 9 using no spaces. The characters "-", "_", "/", "\", "@" and "." are in fact also allowed, but not recommended. The length is restricted to 2-32 characters.

- `cntDefinition`

This is the element holding the configuration of one single counter that is part of the collection.

- `active`

See attribute description in Section 3.3.3 on page 15.

- `activeSnmpp`

See attribute description in Section 3.3.3 on page 15.

- `cntType`

See attribute description in Section 3.3.3 on page 15.

- `identification`

See element description in Section 3.3.3 on page 15.

- `description` (optional)

See element description in Section 3.3.3 on page 15.

- `preferences` (optional)

See element description in Section 3.3.3 on page 15.

- `dataSource`

This is the definition of the data source for the collection.

- `interval`

This attribute defines the interval in seconds between the capturing of PM data operations from the data source.

Possible values: 10, 15, 20, 30, 60, 120, 300, 600, 900, 1200, 1800 and 3600.

Please use the lower values with care. Capturing PM data with a too short interval may overload the ESA as well as the system, which may lead to that the capture time becomes longer than the interval time. Such situation may lead to unexpected PMA behavior and corrupt PM data.



– script

The ESA collects the values returned from the execution of the specified command line. The use of `script` means that the ESA is executing the command line given to the parameter `location`.

It is highly recommended to create the data collection script with some kind of execution control. If the script takes too long time to execute, the PMA will take action. If the script is known to have a long execute time, the solution is to either rewrite the script or select a longer data collection interval.

The PMA has been built with execution and time control. If the script execution time *is longer than half the defined interval*, the ESA will kill the script. At every interval the PMA will continuously trigger the script even though it may be killed due to long execution time. If the script is killed, the counter value is not updated for that interval.

When triggering the script, please be aware of that all environment variables but `TERM`, `PATH` and `LANG` are discarded. This is done by service handling of the PMA service and not by the ESA itself. If you need to use environment variables within the script, they must be sourced within the script.

The ESA collects the values returned by the specified shell script.

`location` The command line to execute, which returns the counter values for the counters in the collection. Script arguments may be used in the command line.

Example 1: Shell script

```
<script>
  <location>/<path>/myScript.sh</location>
</script>
```

Example 2: Java

```
<script>
  <location>/<path>/java -jar getMyCounter.java</location>
</script>
```

Example 3: Perl script

```
<script>
  <location>/<path>/myPerlScript.pl</location>
</script>
```

The response from the command line execution must be in the following format:

```
<timestamp>;<groupid>;<collectionid>
```



```
<counterid1>;<countervalue1>
<counterid2>;<countervalue2>
...
<counteridn>;<countervalueen>
```

timestamp	The timestamp to use for all counters in the collection according to the format <code>YYYYMMDDhhmm[ss]</code> where the seconds <code>ss</code> are optional. If seconds are not specified, the ESA will use <code>hhmm00</code> . The value indicating seconds should be used when the data collection interval defined is less than a minute.
groupid	The group identity to which the collection belong to.
collectionid	The collection identity holding the counters being reported.
counterid	The counter identity being reported.
countervalue	The counter value for the counter being reported.

Example:

```
201409211445;MYGROUP;MYCOLLECTION
MYCNT1;12345
MYCNT2;128
MYCNT3;98765
```

The values returned from the command line execution are mapped to the defined counters within the defined group and collection.

If the return data contain values of counters not defined in the collection, an error is logged.

If the return data does not contain the value of one or several counters in the collection, an error is logged.

Note: The script execution is built with execution and time control. For more information see the script definition description at Page 20.

3.4 Configuration Deployment

The counter definition configuration files are stored in the following directory.

```
{esa basedir}/conf/pmCounters/.
```

The format and syntax of the counter definition configuration file is verified with an XML Schema. The XML Schema file is found at the following location:

```
{esa basedir}/conf/pmCounters/pmCounter.xsd
```



Please note that the counter definition configuration directory is configurable. See Reference [2]. The directories described in this section are the default directories.

3.5 Configuration Activation

At PMA startup the PMA reads all the XML files available in the counter definition directory. If the files and file content are in correct format the content is loaded and activated. If not, the errors found are written to the PMA log file. The PMA is still started, but neglects the erroneous configurations. The PMA also supports runtime loading of configuration files, which means that changes made while the PMA is running will be detected by the PMA and it will reload the configuration without a restart.

3.6 Examples

Counter definition configuration file examples are found in directory:

```
{esa basedir}/conf/examples/
```



4 Define a Job

4.1 Introduction

The purpose of a job is to get the ESA PMA to produce PM output files in 3GPP format. Without a job definition, no files are produced. This simply means that if PM data on files are not needed, there is no need to define jobs.

4.2 Function Description

The job definition in brief is about creating a job with a unique identity, describing which counter group to publish (not a single counter) and, if needed, time statements for start time, stop time and the granularity period. The job configuration also provides the capability to replace the 3GPP XML format with an own *output reporter*, which means the output PM data can be written to a file in a format of own choice or maybe writing the data to a database. This requires creating an own reporter class in java.

4.3 Configuration Format

4.3.1 Overview

The figure below shows a high-level overview of the job definition configuration file.

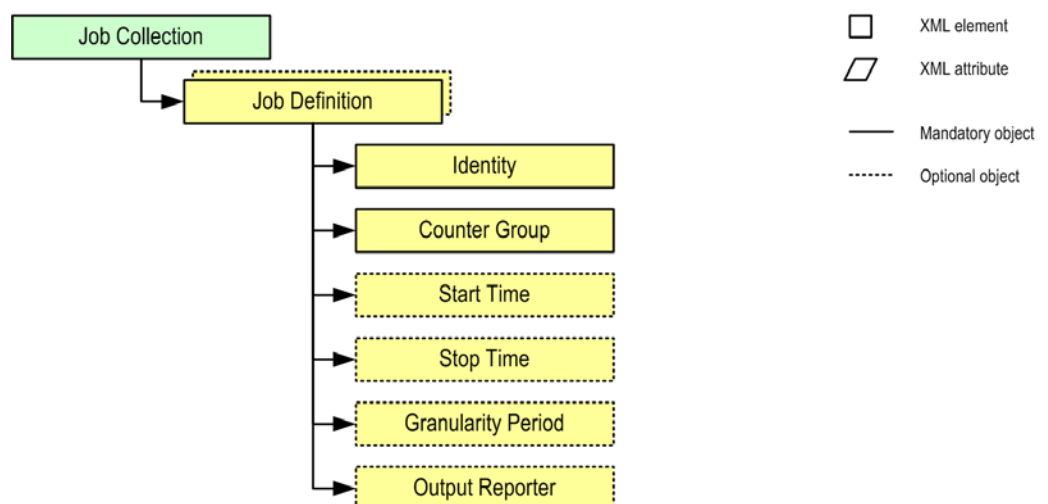


Figure 3 An overview of the XML objects used to create one or several PM jobs in the ESA PMA.

A job definition configuration file contains a collection of job definitions.

Please note that only job identity and a counter group are mandatory parameters. The details of the job definition configuration file are found in the following chapters.

4.3.2 Basic Configuration

The following shows the format of a basic PM job configuration:

```
<jobDefinition active="data">
  <jobId>data</jobId>
  <groupId>data</groupId>
</jobDefinition>
```

The XML element and attribute definitions:

- `jobDefinition`

This element holds the configuration of a job.

- `active`

This attribute is for activating and deactivating the counter. Instead of removing the configuration of a counter when the counter is of no interest, this attribute is simply set to “no” to make it inactive. The result is that the data is not collected, the counter can not be viewed and jobs can not be defined for the counter.

yes	The counter is active.
no	The counter is inactive.

- `jobId`

The job identity. It is a unique identity identifying the job. If there are two or more jobs with the same identity, the second job is not started and an error is logged.

Valid characters are A to Z, a to z and 0 to 9 using no spaces. Also characters “-” and “_” are allowed. The length is restricted to 2-40 characters.

The job identity is used at the following locations:

- 3GPP XML Filename

The job identity is added as is to the filename according to the following format.

A20140921.1500-1505-*<jobId>*_ESA.xml

If the job identity is set to MYJOB, the file name will be the following.



```
A20140921.1500-1505-MYJOB_ESA.xml
```

Please note that using characters "-" or "_" in the job identity string, such as MY-JOB or MY_JOB will lead to the following filenames.

```
A20140921.1500-1505-MY-JOB_ESA.xml
A20140921.1500-1505-MY_JOB_ESA.xml
```

As you can see the characters "-" and "_" might be misleading since there are already such characters used to separate other data in the filename. Please try to avoid using the characters "-" and "_".

Note that the string `ESA` in the example filename is the *PM Unique Identity*, which is configurable and described in Reference [2].

– 3GPP XML Content

The job identity is used within the XML file for identifying which job that created the output XML file. The following is an extraction of an XML output example with job identity `MYJOB`.

```
<measData>
  <managedElement/>
  <measInfo>
    <job jobId="MYJOB"/>
    ...cut...
  </measInfo>
</measData>
```

- `groupId`

The counter group to publish to the output file. All the counters in the group are published. It is not possible to select a set of counters or a single counter, only a group.

The definition of groups is described in Section 3 on page 9.

4.3.3 Customized Configuration

The following shows the format of a customized PM job configuration:

```
<jobDefinition active="data">
  <jobId>data</jobId>
  <groupId>data</groupId>
  <startTime>data</startTime>
  <stopTime>data</stopTime>
  <granularityPeriod>data</granularityPeriod>
  <outputReporterClass>data</outputReporterClass>
</jobDefinition>
```

The XML element and attribute definitions:

- `jobDefinition`

See description in Section 4.3.2 on page 30.

– `active`

See description in Section 4.3.2 on page 30.

- `jobId`

See description in Section 4.3.2 on page 30.

- `groupId`

See description in Section 4.3.2 on page 30.

- `startTime` (optional)

The job start time in format `YYYYMMDDhhmm`.

Example: 201409211200 meaning 21st of September 2014 at 12:00.

If the element is not used, the PMA starts the job immediately when started.

- `stopTime` (optional)

The job stop time in format `YYYYMMDDhhmm`

Example: 201409211800 meaning 21st of September 2014 at 18:00.

If the element is not used, the PMA ends the job only when the PMA itself is stopped.

- `granularityPeriod` (optional)

The time interval in minutes between file outputs. Allowed values are 5, 15, 30 and 60 minutes.

If the element is not used, default value 15 minutes is used.

4.3.4 Advanced Configuration

The following shows the format of an advanced PM job configuration:

```
<jobDefinition active="data">
  <jobId>data</jobId>
  <groupId>data</groupId>
  <outputReporterClass>data</outputReporterClass>
</jobDefinition>
```

or



```
<jobDefinition active="data">
  <jobId>data</jobId>
  <groupId>data</groupId>
  <outputReporterGroup std3gppxml="data">
    <outputReporterClass>data</outputReporterClass>
  </outputReporterGroup>
</jobDefinition>
```

The XML element and attribute definitions:

- `jobDefinition`

See description in Section 4.3.2 on page 30.

– `active`

See description in Section 4.3.2 on page 30.

- `jobId`

See description in Section 4.3.2 on page 30.

- `groupId`

See description in Section 4.3.2 on page 30.

- `outputReporterGroup`

The element holds one or several PM output reporter classes.

- `std3gppxml`

This attribute specifies if 3GPP XML files shall be created for this job. This can be set to "no" when other PM output reporter classes are used.

yes	PM output in 3GPP XML file format is created for this job.
no	PM output in 3GPP XML file format is not created for this job.

- `outputReporterClass` (optional)

The reporter class to use for the PM output reporter.

The PMA comes with an open reporter interface, which means it is possible to create own PM output reporters. This is useful when the output from the PMA is needed in another format than the default 3GPP XML format. See Section 10.5 on page 64, which describes how to create own PM output reporters.

Example reporters that could be created are "Store PM data to CSV files" and "Store PM data in a SQL database".



4.4 Configuration Deployment

The job definition configuration files are stored in the following directory.

```
{esa basedir}/conf/pmJobs/.
```

The format and syntax of the job definition configuration file is verified with an XML Schema. The XML Schema file is found at the following location:

```
{esa basedir}/conf/pmJobs/pmJob.xsd
```

Please note that the job definition configuration directory is configurable. See Reference [2]. The directories described in this section are the default directories.

4.5 Configuration Activation

At PMA startup the PMA reads all the XML files available in the job definition directory. If the files and file content are in correct format the content is loaded and activated. If not, the errors found are written to the PMA log file. The PMA is still started, but neglects the erroneous configurations. The PMA also supports runtime loading of configuration files, which means that changes made while the PMA is running will be detected by the PMA and it will reload the configuration without a restart.

4.6 Examples

Job definition configuration file examples are found in directory:

```
{esa basedir}/conf/examples/
```



5 Define a Threshold

5.1 Overview

The purpose of using thresholds is to send SNMP notifications when the counters breach defined threshold levels.

5.2 Function Description

What must be defined is one or several threshold collections where each contains one or several threshold definitions. The threshold is defined by specifying which counter to monitor and how to monitor it by selecting either type “level” or “span”. Finally, the alarm to trigger when the threshold is breached is also specified. Please note that the alarm to trigger must be defined in the FMA, see Reference [3].

The figure below shows a high-level overview of the threshold definition configuration file.

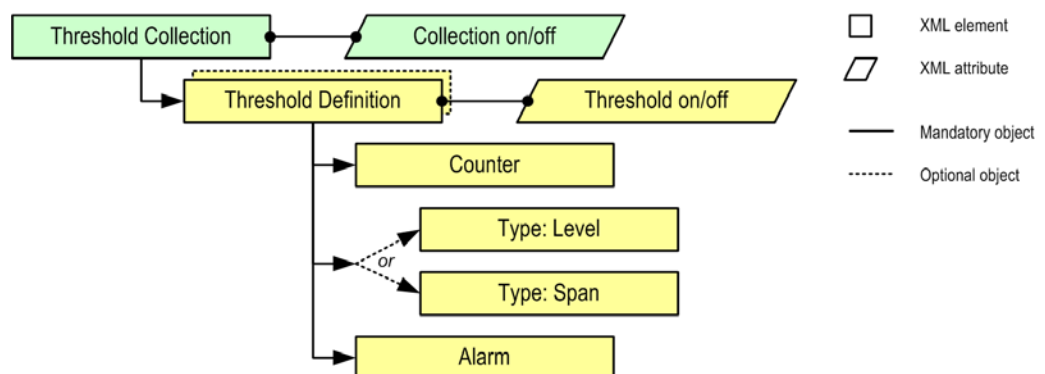


Figure 4 An overview of the XML objects used to create one or several PM thresholds in the ESA PMA.

The box “Counter” represents a single *counter* or a single or multiple *counter instances*. The threshold type specified is valid for all counters or counter instances specified.

The threshold types “level” and “span” are presented in Figure 5. The “level” indicates a level as a limit for what is ok (OK) and not ok (NOK). When the limit is breached an alarm is sent. The “span” has also limits, but there is an upper limit and a lower limit that indicates if the PM counter value is ok or not ok.

Also, there is an attribute in the configuration file allowing to invert the limits defined. Normally a limit is breached when the value rises above a limit, such as when a CPU load “is higher than 90%”. In situations such as monitoring

success rates the goal is to stay at 100%. If the success rate is below 95% an alarm should be sent. This scenario is setup using the invert option.

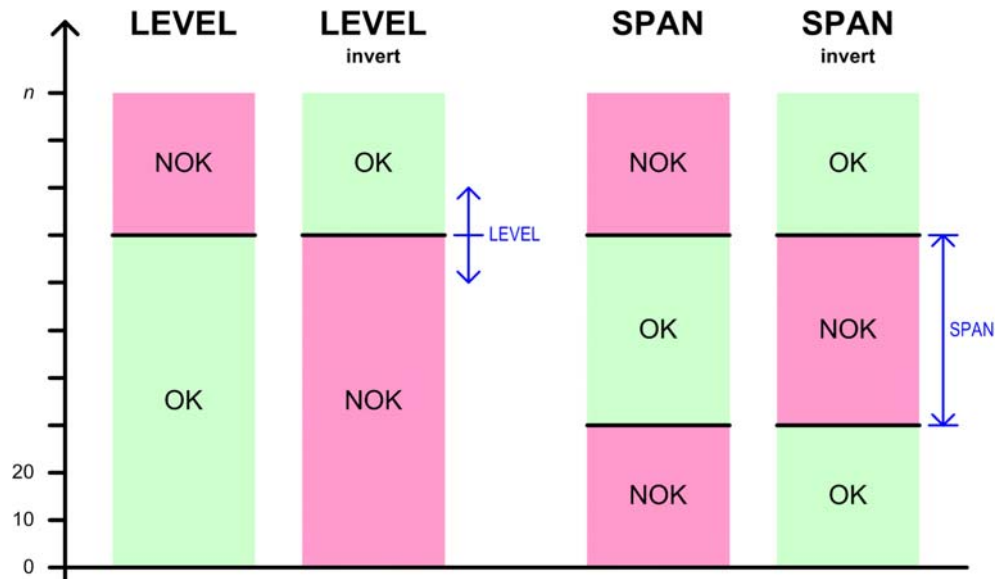


Figure 5 An overview of the threshold types available when threshold monitoring is setup in the PMA.

Even though the Figure 5 presents the limits as a single line, the configuration parameters allows the use of hysteresis; rising and falling limits. The Hysteresis operates as follows:

- Generates a rising alarm when the sample value first becomes greater than or equal to the *rising threshold*. Subsequent rising alarms will only be generated once the sample value becomes less than or equal to the falling threshold.
- Generates a falling alarm whenever the sample value becomes less than or equal to the *falling threshold* and a rising alarm was previously generated.

If the falling threshold is not defined, the falling threshold is the same as the rising threshold.

Figure 6 shows an example of how alarms are triggered when the type “level” is used. When the counter value rises above the *rise* definition an alarm is raised. If the optional *fall* definition is used, the alarm is cleared when the fall level is reached.

If the definition is *inverted*, the fall level becomes the alarm raise level and the rise level becomes the alarm clear level.

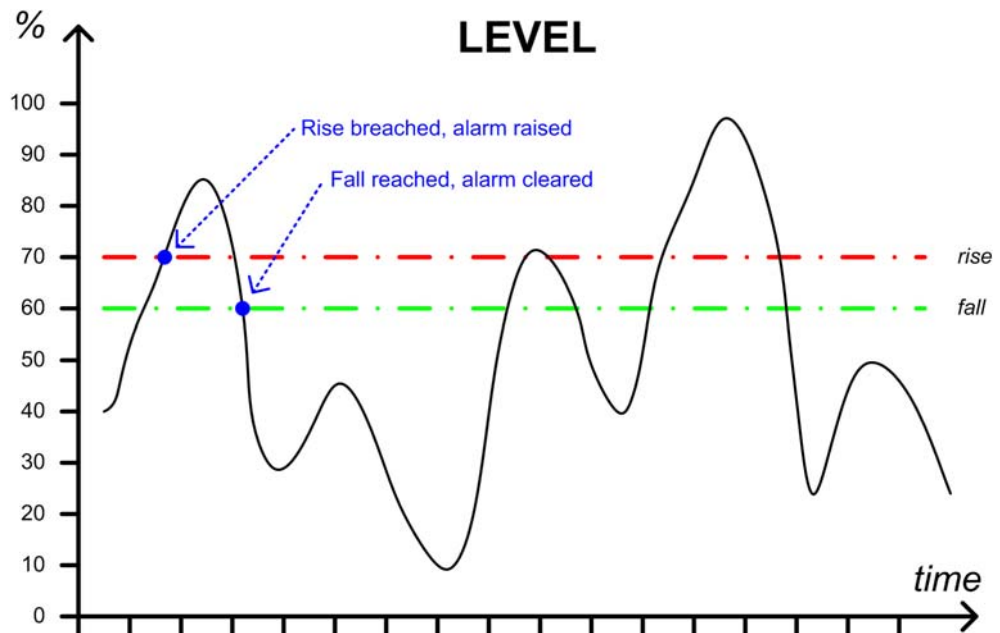


Figure 6 An example of using the threshold type “level” in the PMA.

Figure 7 shows an example of how alarms are triggered when the type “span” is used. When the counter value rises above the *span high* definition an alarm is raised. If the optional *span high return* definition is used, the alarm is cleared when the span high return level is reached. If the span high return level is not defined, the span high return level is set to the same as span high. The same rules applies for the low level definition within the span configuration.

If the definition is *inverted*, the span high/low return level becomes the alarm raise level and the span high level becomes the alarm clear level.

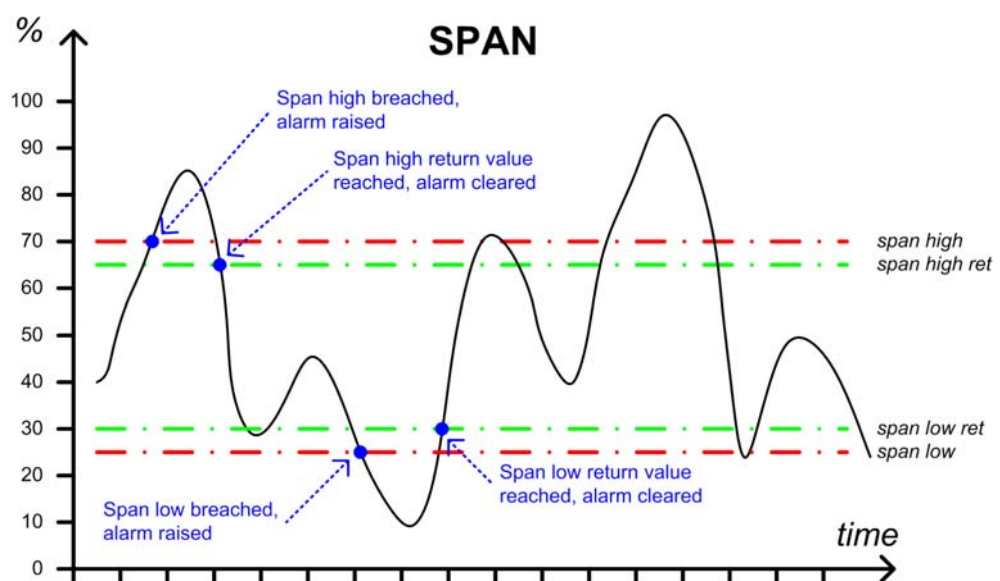


Figure 7 An example of using threshold type “span” in the PMA.



5.3 Configuration Format

5.3.1 Skeleton Format

The following is an example of how to define a PM threshold for a counter:

```
<threshold active="yes">
  <counter>
    <groupId>MYGROUP</groupId>
    <counterId>MYCOUNTER1</counterId>
  </counter>
  <type>

    ...type definition added here...

  </type>
  <output type="alarm">

    ...alarm definition added here...

  </output>
</threshold>
```

The following is an example of how to define a PM threshold for counter instances:

```
<threshold active="yes">
  <instance>
    <groupId>MYGROUP</groupId>
    <counterId>MYCOUNTER1</counterId>
    <instanceId>MYINSTANCE1</instanceId>
    <instanceId>MYINSTANCE2</instanceId>
    <instanceId>MYINSTANCE3</instanceId>
  </instance>
  <type>

    ...type definition added here...

  </type>
  <output type="alarm">

    ...alarm definition added here...

  </output>
</threshold>
```

The XML element and attribute definitions:

- threshold

The configuration of a single threshold definition.



- `counter`

The PM counter to monitor. The definition of groups and counters is described in Section 3 on page 9.

- `groupId`

The counter group that the counter to monitor belongs to.

- `counterId`

The counter to monitor.

- `instance`

The PM counter instance to monitor. The definition of groups, counters and instances is described in Section 3 on page 9.

- `groupId`

The counter group that the counter instance(s) to monitor belongs to.

- `counterId`

The counter to monitor.

- `instanceId`

The counter instance to monitor.

This element is optional. If not specified, all instances of the specified counter are monitored.

- `type`

The monitoring threshold type. Details are described in Section 5.3.2 on page 39.

- `output`

The output action to trigger when threshold is breached is specified.

This is about triggering a defined alarm in the ESA FMA. How to define alarms is described in Reference [3].

The alarm text message can be made flexible and dependent of the threshold situation. How to use the flexible output is described in Section 5.3.3 on page 41.

5.3.2 Type Format

The following are examples of how to define PM threshold types:

```
...
<type>
  <level invert="no">
    <rise>80</rise>
    <fall>70</fall>
  </level>
</type>
...

or

...
<type>
  <span invert="no">
    <high>60</high>
    <highRet>55</highRet>
    <lowRet>45</lowRet>
    <low>40</low>
  </span>
</type>
...
```

The XML element and attribute definitions:

- type

The element holding the monitoring threshold type definition.

- level

The element holds the definition of monitoring threshold type “level”.

– invert

Inverts the defined level limits.

yes Defined levels are inverted.

no Defined levels are not inverted.

– rise

When the value of the PM counter is equal to or rises above the value of this element, an alarm raise is triggered. If inverted, this element triggers an alarm clear.

– fall

When the value of the PM counter is equal to or falls below the value of this element, an alarm clear is triggered. If inverted, this element triggers an alarm raise.

- span



The element holds the definition of monitoring threshold type “span”.

- `invert`

Inverts the defined span limits.

`yes` Defined levels are inverted.

`no` Defined levels are not inverted.

- `high`

When the value of the PM counter is equal to or rises above the value of this element, an alarm raise is triggered. If inverted, this element triggers an alarm clear.

- `highRet`

When the value of the PM counter is equal to or falls below the value of this element, an alarm clear is triggered. If inverted, this element triggers an alarm raise.

- `lowRet`

When the value of the PM counter is equal to or rises above the value of this element, an alarm clear is triggered. If inverted, this element triggers an alarm raise.

- `low`

When the value of the PM counter is equal to or falls below the value of this element, an alarm raise is triggered. If inverted, this element triggers an alarm clear.

5.3.3 Output Format

The alarm to send when a threshold is breached is defined according to descriptions in Reference [3]. However, the text that is part of the alarm is made flexible and dynamic by adding codes that are related to the actual threshold breach.

The following are examples of a defined alarm to send when a threshold is breached:

Format A) Same alarm texts for raise alarm and clear alarm

```
...
<output type="alarm">
  <moduleId>PERFORMANCE</moduleId>
  <errorCode>12</errorCode>
  <resourceId>1.1.1.2.3</resourceId>
  <text>Level threshold breached.</text>
```

```

    <origSourceIp>10.21.9.72</origSourceIp>
</output>
...

```

Format B) Differentiated texts for raise alarm and clear alarm

```

...
<output type="alarm">
  <moduleId>PERFORMANCE</moduleId>
  <errorCode>12</errorCode>
  <resourceId>1.1.1.2.3</resourceId>
  <textGrp>
    <textRaise>Threshold breached.</textRaise>
    <textClear>Threshold returned back to normal.</textClear>
  </textGrp>
  <origSourceIp>10.21.9.72</origSourceIp>
</output>
...

```

The XML element and attribute definitions:

- output

The element holds the SNMP message definition.

- type

The SNMP notification type to send.

alarm	When a breach occur an <i>alarm raise</i> is sent and when a breach is resolved an <i>alarm clear</i> is sent.
-------	--

event	An <i>event</i> is sent both when a breach occurs and when it is resolved.
-------	--

- moduleId

The alarm module identification.

- errorCode

The alarm error code identification.

- resourceId

The alarming object resource identity.

- text | textGrp

Optional elements holding the text to include in the alarm sent from the ESA.

If these elements are not specified, the default alarm text defined in the alarm definition configuration is used.



<code>text⁽¹⁾</code>	The text is used both when the threshold is breached and when the threshold returns back to normal.
<code>textRaise</code>	The text is used when the threshold is breached.
<code>textClear</code>	The text is used when the threshold returns back to normal.

(1) This option is kept for backward compatibility. It is recommended to use `<textRaise>` and `<textClear>` though.

- `origSourceIp`

An optional element that specifies the IP address of the alarm being triggered.

If the element is not specified, the ESA uses the IP address where the PM Agent is running.

The elements `text`, `textRaise` and `textClear` can include codes representing dynamic data that are added to the text string at the trigger time of an alarm. The following codes may be used:

@ig@	The identity of the group that the monitored counter belongs to.
@ic@	The identity of the counter being monitored.
@ii@	The identity of the instance being monitored.
@cv@	Current value of the counter. In other words, this is the value that breached the threshold definition.
@tlr@	A type “level” threshold and value for “rise”.
@tlf@	A type “level” threshold and value for “fall”.
@tsh@	A type “span” threshold and value for the high limit.
@tshr@	A type “span” threshold and value for the high return limit.
@tslr@	A type “span” threshold and value for the low return limit.
@tsl@	A type “span” threshold and value for the low limit.

Example text string using dynamic data:

```
...
<text>Counter @ic@ in group @ig@ breached
      level @tlr@ (current value is @cv@).</text>
...
```

may lead to an alarm text looking like:

```
Counter SESSIONS in group MSGPROC breached  
level 500 (current value is 508).
```

5.4 Configuration Deployment

The threshold definition configuration files are stored in the following directory.

```
{esa basedir}/conf/pmThresholds/.
```

The format and syntax of the threshold definition configuration file is verified with an XML Schema. The XML Schema file is found at the following location:

```
{esa basedir}/conf/pmThresholds/pmThreshold.xsd
```

Please note that the threshold definition configuration directory is configurable. See Reference [2]. The directories described in this section are the default directories.

5.5 Configuration Activation

At PMA startup the PMA reads all the XML files available in the threshold definition directory. If the files and file content are in the correct format the content is loaded and activated. If not, the errors found are written to the PMA log file. The PMA is still started, but neglects the erroneous configurations. The PMA also supports runtime loading of configuration files, which means that changes made while the PMA is running will be detected by the PMA and it will reload the configuration without a restart.

5.6 Examples

Threshold definition configuration file examples are found in directory:

```
{esa basedir}/conf/examples/
```




6 Output SNMP

6.1 Introduction

The PMA output to SNMP works according to the following guide:

- Each PMA counter definition has an option to present its counter value over SNMP or not.
- If the SNMP option is set to “no” the counter is not visible on SNMP even though the counter is active and visible in the PMA. The OSS can not read the counter value.
- If the SNMP option is set to “yes” the counter is visible on SNMP. The OSS can read the counter value.
- The SNMP table holding the counters is indexed with GROUPID and COUNTERID.
- The SNMP view shows the counter value only. No historical values are presented.

6.2 SNMP MIB Format and Content

The SNMP MIB format and content is in more detail described in Reference [4].





7 Output 3GPP XML File

7.1 Introduction

The PMA output to XML file works according to the following guide:

- The PMA creates PM result files depending on the counter definition configuration as well as the jobs defined and in progress. If there are no jobs defined, there are no PM result files produced.
- A defined job contains a PM group selection, which means all the counters within the selected group is part of a PM result file as the result of a job.
- One PM result file is created once within each granularity period interval until the job is removed. Jobs never append data to any PM result file.
- Activating file retention may lead to that PM result files are deleted when getting too old.

See file retention details in Reference [2].

7.2 File Format and Content

The 3GPP XML file format and content is in more detail described in Reference [5].





8 Administration

8.1 Log Files

The log files produced by the ESA PM Agent are described in Reference [2].

8.2 Backup and Restore

Backup of the ESA configuration, including the PM Agent configuration files, is described in Reference [2].





9 Command Line Interface

9.1 Introduction

During runtime the PM Agent will contain counters and counter values and the data is made available to the OSS using SNMP and 3GPP XML files. But, also the technician working locally with the system could benefit from looking at the data captured and produced by the PM Agent.

A number of CLI commands exist that makes the setup and configuration of the PM Agent easier. Also, they are useful when troubleshooting faults in the system by looking at the PM counters and PM threshold alarms.

The following CLI commands are available for the PM Agent:

- ☐ `pmcounterstatus`
- ☐ `pmresetcounter`
- ☐ `pmverifydatasource`
- ☐ `pmreadcounter`
- ☐ `pmjob`
- ☐ `pmwritecounter`
- ☐ `pmthresholdalarms`

9.2 Command: Counter Status

9.2.1 Description

This command is used for checking the counter status. The command will return the status information indicating counter active or inactive.

9.2.2 Command Format

The command format:

```
pmcounterstatus <groupId> <counterId>
```

The command attributes are the following:

groupId	The group identity of the group that the counter belongs to.
----------------	--



counterId The counter identity of the counter to check.

9.2.3 Command Output

The command gives the following output:

ACTIVE The counter is defined, loaded and active in the PM Agent.

INACTIVE The counter is defined and loaded, but made inactive, in the PM Agent.

If the counter specified does not exist in the counter definition configurations, the command gives the following output:

UNDEFINED The counter is not defined in the PM Agent.

9.3 Command: Reset Counter

9.3.1 Description

This command is used for resetting a counter value to its reset value. The reset value is entered in the PM Counter Definition configuration file. If there is no reset value given, the command will return an error message.

9.3.2 Command Format

The command format:

```
pmresetcounter <groupId> <counterId>
```

The command attributes are the following:

groupId The group identity of the group that the counter belongs to.

counterId The counter identity of the counter to reset.

9.3.3 Command Output

If the counter is successfully reset, the command does not give any output.

If the counter could not be reset, the command returns an error message.



9.4 Command: Verify Data Source

9.4.1 Description

This command is used for verifying that the defined data source for a counter or collection exists, is valid and working. The command should be used on the counter before including the counter in the PM Agent counter definition configuration directory.

Please note that this command can be executed with the PM Agent being shutdown.

9.4.2 Command Format

The command format:

```
pmverifydatasource <groupId> <counterId> <counterDefFile>
```

The command attributes are the following:

groupId	The group identity of the group that the counter belongs to.
counterId	The counter identity of the counter data source to verify.
counterDefFile	The counter definition configuration XML file that contains the data source to verify.

9.4.3 Command Output

The command output shows the result from capturing the PM data from the data source itself. If it is verified that the command output shows the same value as what is found in the data source, the counter data source is correct.

In the examples below there is one counter collection MYCOLL with two counters named MYCNT1 and MYCNT2 within group MYGRP being defined in XML file cntdef.xml.

The following table holds example values for the counters in the example.

Table 1

Group	Collection	Counter	Value
MYGRP	MYCOLL	MYCNT1	1234
MYGRP	MYCOLL	MYCNT2	2222

Example 1:

```
# pmverifydatasource MYGRP MYCNT1 cntdef.xml
MYGRP; MYCNT1; 1234
```

**Example 2:**

```
# pmverifydatasource MYGRP MYCNT2 cntdef.xml
MYGRP; MYCNT2; 2222
```

9.5 Command: Read Counter

9.5.1 Description

This command is used for viewing counter information and values and can be used to read both stored values (last read values) from the PM Agent and live values (real time values) being read from the actual data source.

9.5.2 Command Format

The command format:

```
pmreadcounter [<groupId> <counterId> [<options>]]
```

The command attributes are the following:

groupId	The group identity of the group that the counter belongs to.
counterId	The counter identity of the counter to view.
options	The level of detail in the command output.
-d	Counter description.
-l	Counter live value. The value is fetched from the data source at the execution of the command. This value does not update the value of the counter.
-t	Counter timestamp. This indicates when the counter was last updated.
-v	Counter value.

9.5.3 Command Output

In the output examples below there are two values used. A *stored value*, which is the value currently processed by the PMA, and a *live value*, which is the value that is taken from the data source at the time of running the command. The following values are used in the examples:

- Stored value @ 2014-09-21 12:10:00 is 3572.
- Live value @ 2014-09-21 12:13:14 is 4242.



The command provides two output formats.

- **Raw output**

This output is received when no command options are given. The output is a table showing the counters, the timestamps and the values.

- **Customized output**

This output is received when one or several command options are given. This is a limited output which only shows what is specified by the user.

The raw output is a list of counters with a header and a default set of columns separated with semicolons.

Example R1:

The following example prints all active counters in the PM Agent.

```
# pmreadcounter
Group id; Counter id; Instance id; Timestamp; Counter
Value
<Group Id 1>; <Counter Id 1>; <Instance Id 1>; <Timestamp 1>;
<Value 1>
<Group Id 2>; <Counter Id 2>; <Instance Id 2>; <Timestamp 2>;
<Value 2>
<...>
<Group Id n>; <Counter Id n>; <Instance Id n>; <Timestamp n>;
<Value n>
```

Example R2:

The following example prints all instances of the abstract counter MYCOUNTER in counter group MYGROUP in the PM Agent.

```
# pmreadcounter MYGROUP MYCOUNTER
Group id; Counter id; Instance id; Timestamp; Counter
Value
MYGROUP; MYCOUNTER; <Instance Id 1>; <Timestamp 1>; <Value 1>
MYGROUP; MYCOUNTER; <Instance Id 2>; <Timestamp 2>; <Value 2>
<...>
MYGROUP; MYCOUNTER; <Instance Id n>; <Timestamp n>; <Value n>
```

The customized output allows the user to filter out a single counter or a set of counters as well as get only specific counter data instead of the default ones by using command options.

Example C1:

```
# pmreadcounter MYGRP MYCNT -t -v
2014-09-21 12:10:00; 3572
```

Example C2:

```
# pmreadcounter MYGRP MYCNT -v -l
3572; 4242
```

Example C3:



```
# pmreadcounter MYGRP MYCNT -t -v -l  
2014-09-21 12:10:00; 3572; 2014-09-21 12:13:14; 4242
```

Example C4:

```
# pmreadcounter MYGRP MYCNT -d -l  
Group: Feature Message Control  
Counter: Number of Transactions per minute  
4242
```

Example C5:

```
# pmreadcounter MYGRP MYCNT -d -l -t  
Group: Feature Message Control  
Counter: Number of Transactions per minute  
2014-09-21 12:13:14; 4242
```

Example C6:

```
# pmreadcounter MYGRP MYCNT -d -t -l -v  
Group: Feature Message Control  
Counter: Number of Transactions per minute  
2014-09-21 12:10:00; 3572; 2014-09-21 12:13:14; 4242
```

9.6 Command: Write Counter

9.6.1 Description

This command is used for updating a counter with a new counter value.

9.6.2 Command Format

The command format:

```
pmwritecounter <groupId> <counterId> [<instanceId>] <cntValue>  
[<timestamp>]
```

The command attributes are the following:

groupId	The group identity of the group that the counter belongs to.
counterId	The counter identity of the counter to update.
instanceId	The instance identity of the instance to update.
cntValue	The value to set for the counter or instance.
Datatype: Integer	



timestamp The timestamp to use for the setting of counter value. If timestamp is not given, the timestamp is set to the timestamp for the operation.

Datatype: Date, 'YYYY-MM-DD hh:mm:ss'

9.6.3 Command Output

If the counter is successfully updated, the command does not give any output.

If the counter could not be updated, the command returns an error message.

9.7 Command: View Job

9.7.1 Description

This command is used for viewing counter jobs that are configured and managed in the PM Agent.

Please note that the command does not allow changes to be made to the jobs. The job configuration is done by editing the job definition configuration files. The command is a view into the ESA to show which jobs the ESA are processing.

9.7.2 Command Format

The command format:

```
pmjob [<jobId>] [<options>]
```

The command attributes are the following:

jobId The job identity of the job to view.

**options**

The level of detail in the command output.

- v Show all job details.
- s Show jobs with a job state. This option requires one of the following states:
 - `inactive(i)`
Show all jobs that are configured, but made inactive.
 - `waiting(w)`
Show all jobs that are waiting to get started.
 - `running(r)`
Show all jobs that are in operation.
 - `stopped(s)`
Show all jobs that are finalized and ended.

9.7.3**Command Output**

The command provides two output formats.

- List output
- Detailed output

The list output is a list of jobs with a header and columns separated with semicolons.

```
Job id; Group id; Job State; Next Reporting Time
<Job Id 1>;<Group Id 1>;<Job State 1>;<Next Reporting Time 1>
<Job Id 2>;<Group Id 2>;<Job State 2>;<Next Reporting Time 2>
...
<Job Id n>;<Group Id n>;<Job State n>;<Next
Reporting Time n>
```

The detailed output is a block of data for the job.

```
!-----
Job id           : <Job Id>
Group id        : <Group Id>
Job State       : <Job State>
Start Time      : <Start Time>
Stop Time       : <Stop Time>
Granularity Period : <Granularity Period>
Standard 3GPP XML : <3gpp Xml Output>
Output Reporter Class :
    <Status 1> <Output Reporter Class 1>
    <Status 2> <Output Reporter Class 2>
    ...
    <Status n> <Output Reporter Class n>
```



```
Next Reporting Time      : <Next Reporting Time>
-----!
```

Example 1:

```
# pmjob
Job id; Group id; Job State; Next Reporting Time
JOBABC; GROUPSYSRES; RUNNING; 2014-05-05 05:05:00
JOBSYSTEM; GROUPDATABASE; WAITING; 2020-02-20 20:00:00
JOBXYZ; GROUPLHW; INACTIVE;
JOBTEST; GROUPAPPL; STOPPED;
```

The example 1 shows that the ESA is maintaining four jobs, but only two are active. The third job in the list is intentionally made inactive and the fourth job in the list is finished as the configured stop time has been reached. The second job in the list is waiting to be started as it will start when the configured start time is reached.

Example 2:

```
# pmjob -s r
Job id; Group id; Job State; Next Reporting Time
JOBABC; GROUPSYSRES; RUNNING; 2014-05-05 05:05:00
```

The example 2 shows all jobs that are maintained by the ESA and in state RUNNING.

Example 3:

```
# pmjob JOBABC -v
!-----
Job id           : JOBABC
Group id         : GROUPSYSRES
Job State        : RUNNING
Start Time       :
Stop Time        :
Granularity Period : 5
Standard 3GPP XML : YES
Output Reporter Class :
Next Reporting Time : 2014-04-08 12:05:00
-----!
```

The example 3 shows the details of job JOBABC, which reports counter values for the counter group GROUPSYSRES. The job is currently running and the next reporting time is 2014-04-08 12:05:00. The printout indicates granularity period 5 minutes, which means the job period started at 2014-04-08 12:00:00. There is no customized reporter class configured. The output will be default 3GPP XML Files.

Example 4:

```
# pmjob JOBSYSTEM -v
!-----
Job id           : JOBSYSTEM
Group id         : GROUPDATABASE
Job State        : WAITING
```



```

Start Time           : 2020-02-20 19:00:00
Stop Time            : 2020-02-30 12:00:00
Granularity Period   : 60
Standard 3GPP XML    : YES
Output Reporter Class :
    [ACTIVE] com.company.myagent.reporter.ExampleReporter
    [INACTIVE] com.company.myagent.reporter.DummyReporter
Next Reporting Time   : 2020-02-20 20:00:00
-----!

```

The example 4 shows the details of job JOBSYSTEM, which reports counter values for the counter group GROUPDATABASE. The job is currently in waiting state since the job start time is set in the future, being 2020-02-20 19:00:00. With the granularity period of 60 minutes it means that the next reporting time is 2020-02-20 20:00:00, which is also stated in the printout. The job will automatically end after 10 days, meaning at 2020-02-30 12:00:00. The output will be default 3GPP XML files, but there are also customized reporter classes specified. One output reporter class is configured to be inactive though.

9.8 Command: View Threshold Alarms

9.8.1 Description

This command is used for viewing the active alarms that have been triggered by the PM Agent, which means the alarms that are triggered by getting PM thresholds being breached.

9.8.2 Command Format

The command format:

```
pmthresholdalarms
```

The command does not take any arguments.

9.8.3 Command Output

If there are no active PM alarms, the command output is only a heading.

```
# pmthresholdalarms
Active alarms:
```

If there are active alarms, the command output is a complete list of all active alarms. The output example below shows two active PM threshold alarms. The first one is a *level threshold* alarm and the second is a *span threshold* alarm.

```
# pmthresholdalarms
```




Active alarms:

```
!-----!
Module      : SYSRES
Error Code  : 1001
Resource Id  : 1.1.1.2.8
Timestamp   : Sun Sep 21 09:19:44 CEST 2014
Alarm Text   : CPU Load high. Breached level 90%. 92%
               reported.
Group Id     : SYSRESGRP
Counter Id   : CPULOAD
Current value : 95
Trigger value : 92
Limits       : alarm - 90 | 85 - clear
!-----!
Module      : SYSRES
Error Code  : 2001
Resource Id  : 1.1.1.2.6
Timestamp   : Sun Sep 21 14:14:44 CEST 2014
Alarm Text   : Memory usage high, warning. Breached 70%. 74%
               reported.
Group Id     : MYGROUP
Counter Id   : MYAPICOUNTER
Current value : 79
Trigger value : 74
Limits       : clear - 92 | 90 - alarm - 70 | 68 - clear
!-----!
```





10 Application Programming Interface

10.1 Introduction

The PM API is used by java applications that need to handle counters. It contains the following interfaces and functions:

- **Counter**
- **Job**
- **Threshold**
- **Output Reporter**

The following chapters describe the interfaces in more detail.

10.2 Interface: Counter

This interface gives the API user the possibility to manage PM counters from Java code. The API supports counter operations, such as reading counters and updating counters.

Interface: `com.ericsson.esa.pmagent.rmi.ICounter`

See javadoc in file `esa-javadoc.jar` located in `{esa basedir}/lib`.

Find code examples in file `esa-sources.jar` located in `{esa basedir}/lib`.

10.3 Interface: Job

This interface gives the API user the possibility to manage PM jobs from Java code. The API supports job operations, such as reading job definitions.

Interface: `com.ericsson.esa.pmagent.rmi.IJob`

See javadoc in file `esa-javadoc.jar` located in `{esa basedir}/lib`.

Find code examples in file `esa-sources.jar` located in `{esa basedir}/lib`.

10.4 Interface: Threshold

This interface gives the API user the possibility to manage PM thresholds from Java code. The API supports threshold operations, such as checking alarm status.

Interface: `com.ericsson.esa.pmagent.rmi.IThreshold`

See javadoc in file `esa-javadoc.jar` located in `{esa basedir}/lib`.

Find code examples in file `esa-sources.jar` located in `{esa basedir}/lib`.

10.5 Interface: Output Reporter

This interface gives the API user the possibility to replace the default 3GPP XML output format to an own customized output format. A few examples that could be wanted are "Store PM data in CSV files" and "Store PM data in a SQL database". The ESA itself does not need any software modifications. This is setup with configuration only.

Interface: `com.ericsson.mmas.pm.reporters.Reporter`

See javadoc in file `esa-javadoc.jar` located in `{esa basedir}/lib`.

Find code examples in file `esa-sources.jar` located in `{esa basedir}/lib`.

The ESA user must create a reporter class in java and the java class is added to the PMA job configuration. Instead of using the default 3GPP XML output, the PMA will send its output to the new reporter class and the reporter class will publish the output PM data in any way wanted. See Section 4.3 on page 29, which shows how to configure the PM job to use an own reporter.

Keep in mind that using this customized reporter option makes the ESA loose control of the output data. The following must be handled by the reporter class.

- File target

The target directory is not known by the ESA. The reporter must have its own configuration for defining the target directory.

- File retention

The file retention control in the ESA does not work as the ESA does not know where the output is stored and not in which format.

- File name

The file naming convention used for the 3GPP XML is no longer valid.

- File content

The file content format used for the 3GPP XML is no longer valid.

In order to get the ESA PMA to load the reporter class the classpath must be updated with information about where to find the class file. This is done by editing the PMA `esapma.vmoptions` file.



The following procedure describes all the steps needed to link a new reporter class to the ESA and how to get it into operation.

- 1 Stop the ESA PM Agent.
- 2 Put the reporter class on the file system.

The recommended location is `{esa basedir}/lib`.

- 3 Open file `{esa basedir}/bin/esapma.voptions` for editing.
- 4 Add the following line.

```
-classpath/a {path}/{filename}.jar
```

Example)

```
-classpath/a /opt/esa/lib/MyPmReporter.jar
```

- 5 Save the file.
- 6 Create a job configuration that uses the reporter class.

Example)

```
<jobDefinition active="yes">
  <jobId>MYJOB</jobId>
  <groupId>MYCNTGROUP</groupId>
  <outputReporterClass>
    com.ericsson.myreporter
  </outputReporterClass>
</jobDefinition>
```

- 7 Save the job configuration file named `{filename}.xml` in `{esa basedir}/conf/pmJobs`.
- 8 Start the ESA PM Agent.





Glossary

Glossary

ESA Glossary of Terms and Acronyms,
0033-CSH 109 532





Reference List

- [1] *ESA Library Overview*
DIRECTIONS FOR USE, 1/1553-CSH 109 532
- [2] *ESA Setup and Configuration*
SYSTEM ADMINISTRATION GUIDE, 1/1543-CSH 109 532
- [3] *ESA Fault Management*
SYSTEM ADMINISTRATION GUIDE, 2/1543-CSH 109 532
- [4] *ESA SNMP Interface for Performance Management*
INTERFACE DESCRIPTION, 2/155 19-CSH 109 532
- [5] *ESA XML Interface for Performance Management*
INTERFACE DESCRIPTION, 3/155 19-CSH 109 532