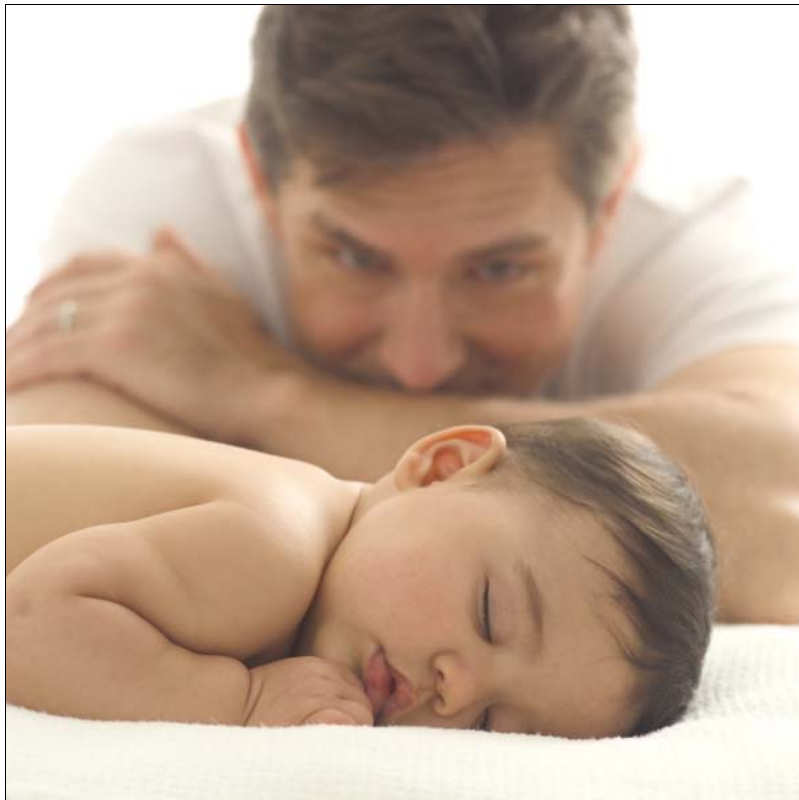


ESA Fault Management

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	Fault Management Overview	3
3	Alarm Definition	5
3.1	Introduction	5
3.2	Function Description	5
3.3	Configuration Overview	11
3.4	Configuration Format	12
3.5	Configuration Deployment	17
3.6	Configuration Activation	17
3.7	Examples	17
4	Alarm Translation	19
4.1	Introduction	19
4.2	Function Description	20
4.3	Configuration Overview	23
4.4	Configuration Format	24
4.5	Configuration Deployment	33
4.6	Configuration Activation	34
4.7	Examples	34
5	Command Line Interface	39
5.1	Introduction	39
5.2	Command: View Active Alarms	39
5.3	Command: Send Message	42
6	Application Programming Interface	45
6.1	Introduction	45
6.2	Interface: Alarm Definition	45
6.3	Interface: Alarm	45



6.4	Interface: Active Alarm List	46
6.5	Interface: Subagent AAL Synchronization	46
7	Administration	49
7.1	Log Files	49
7.2	Backup and Restore	49
8	Appendix A: Standardized Parameter Values	51
8.1	Perceived Severity	51
8.2	Event Type	51
8.3	Probable Cause	52
	Glossary	59
	Reference List	61



1 About This Document

1.1 Purpose

The purpose of this document is to describe the system administration tasks for setting up the Fault Management features in the Ericsson SNMP Agent (ESA).

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.
- Is familiar with Java regular expressions.
- 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 Fault Management Overview

To start using the Fault Management (FM) features of the ESA, configuration of site and system specific alarms and events must be done. This implies definition of the possible alarms and events that can be sent from the system, as well as, if needed, translation of subagent alarms. The latter is required only if one or several SNMP agents are present on the system and if they shall be integrated with the ESA. See Figure 1, which visualizes the usage of alarm definitions and alarm translations.

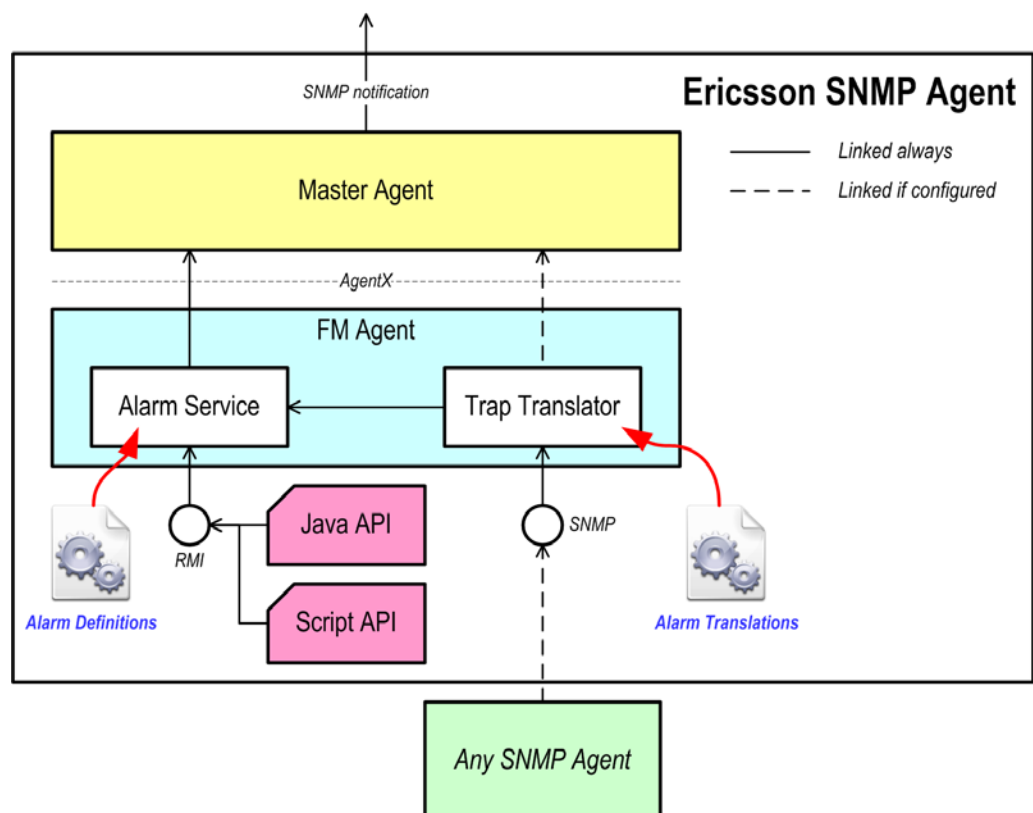


Figure 1 An overview of the FMA configuration, which means both Alarm Definition and Alarm Translation configurations.

The configuration operations are the following.

☐ Alarm Definition

This is about configuring the ESA with alarm definitions for the system specific alarms and events possible to send from the system. The ESA can only send alarms that are specified in the alarm definition configurations.

☐ Alarm Translation



This is about configuring the ESA to intercept and translate SNMP alarms in any SNMP format to a common SNMP format, the ESA format.

The Figure 1 shows where the alarm definitions and alarm translations are utilized in the ESA.

When a sub agent is sending alarms/events and these are to be translated to the ESA format, they are sent to the FM Agent and the Trap Translator (TT) feature. The TT uses the configuration file(s) to map the intercepted alarm to an alarm definition. The Alarm Service (AS) feature takes care of the formatting of the triggered alarm and the Master Agent (MA) is responsible for sending the alarm to the defined trap destination(s).

The picture also shows that the AS may be triggered to send an alarm also by the Java API and the Shell Script interface, which means that the TT is not always used to send alarms and that alarms/events may be triggered directly in the AS.



3 Alarm Definition

3.1 Introduction

The alarm definition is about setting up alarms in a unique manner. The alarms defined are the only alarms possible to send using the ESA. In other words; if the alarm is not defined, it can not be sent.

When defined, they can easily be triggered from the ESA internal processes, using the Java API or the shell script support, as well as through the Trap Translation and PM Agent features.

The alarms are easily defined in one or several XML files.

3.2 Function Description

3.2.1 Overview

The Fault Management concepts, features and functions in the ESA are the following:

- Alarm Format
- Alarm Definition
- Active Alarm
- Send Alarm
- Send Event
- Active Alarm List
- Alarm Cleared List
- Event History List

3.2.2 Alarm Format

The SNMP notifications sent from the ESA are following the MIBs *ERICSSON-SNF-ALARM-MIB* and *ERICSSON-SNF-EVENT-MIB*. There are three SNMP notifications defined and used.

- Alarm Raise
- Alarm Clear

- Event

The alarm message is stateful, which means it is in state active when raised and canceled when cleared. The event message is stateless. Both message types alarm and event contain the same data.

For more information about the SNMP interface for FM, see Reference [3].

3.2.3

Alarm Definition

An alarm is defined in the ESA in order to provide the ESA with information about alarms that can be sent. The defined alarms in the ESA can be read using SNMP according to the MIB *ERICSSON-SNF-MODEL-MIB*. The alarms are defined with the following data.

- Module Identity
- Error Code
- Severity
- Model Description
- Active Description
- Event Type
- Probable Cause

The Module Identity is an identity for a *group of alarms*. One Module can hold multiple alarms. The Error Code is a unique number within the Module Identity that is seen as an *alarm identity*. The concatenated identities of MODULE and ERRORCODE must be a unique definition of a single alarm in the ESA.

**EXAMPLE)**

There are three hardware alarms to be defined. They are triggered when the CPU Load is too high, when the Disk usage is too high and when the Memory usage is too high. The following alarms are defined:

Module id	Error code	Alarm slogan
-----	-----	-----
HARDWARE	101	CPU Load high.
HARDWARE	102	Disk usage high.
HARDWARE	103	Memory usage high.

Also, two database alarms are created. One is to be triggered when the table space usage is too high and the second one when the database process is down. The following alarms are defined:

Module id	Error code	Alarm slogan
-----	-----	-----
DATABASE	101	Table space usage high.
DATABASE	102	Database process down.

As seen in the example above the error codes can be reused as long as the module identity is not the same.

For more information about the SNMP interface for FM, see Reference [3].

3.2.4 Active Alarm

3.2.4.1 General

An alarm is active when it was triggered as an *alarm raise* message. This means that the alarm is raised, but also it is active until it is cleared. It is cleared by sending an *alarm clear* message.

The active alarms are found in the Active Alarm List (AAL). They are found in the AAL as long as they are active and the ESA is not restarted. At ESA restart they are lost.

The AAL holds only one alarm of each kind. If an alarm with the same content is sent twice, the AAL will notice the alarm is already active in the AAL and not put a second one in there. The attributes that make an alarm unique in the AAL are the following:

- Module Identity
- Error Code
- Resource Identity
- Originating Source



The *Resource Identity* identifies the alarming resource. This means that the same alarm can be triggered from different system resources. The resource identity makes the active alarms unique if they are sent from two or more different alarming resources.

EXAMPLE)

One alarm is defined with the alarm slogan "Process down" and the system contains five processes. The following alarm is defined.

Module id	Error code	Alarm slogan
PROCESS	101	Process down.

The following resources are identified.

Resource id	Resource name
1.1.1.20.1	Process A
1.1.1.20.2	Process B
1.1.1.20.3	Process C
1.1.1.20.4	Process D
1.1.1.20.5	Process E

When the processes A, C and D is down the AAL in the ESA will hold the following alarms.

PROCESS:101:1.1.1.20.1	(Proc A)
PROCESS:101:1.1.1.20.3	(Proc C)
PROCESS:101:1.1.1.20.4	(Proc D)

The same alarm is triggered three times, but since the resource identity is different the AAL will hold three alarms.

Let's assume that process C is up and running. The alarm is cleared. This will lead to the following AAL content.

PROCESS:101:1.1.1.20.1	(Proc A)
PROCESS:101:1.1.1.20.4	(Proc D)

The *Originating Source* identifies the alarming source, such as an IP address. This means that the same alarm triggered from different sources is treated as multiple alarms. The originating source makes the active alarms unique if they are sent from two or more different alarming sources. Normally the ESA is used stand-alone in a node, which means all alarms are triggered from the same source, which means all alarms have the same Originating Source IP. However, when the ESA is integrated with external entities with different IP addresses and/or when the ESA runs in Cluster Mode, the ESA will have alarms with multiple Originating Source IPs. To clear an alarm, the clearing



must be executed on the node that triggered the alarm or the Originating Source IP address must be given in the clear operation.

EXAMPLE)

One alarm is defined with alarm slogan "CPU Load high" and the system contains five nodes (one IP address each). The following alarm is defined in all five nodes.

Module id	Error code	Alarm slogan
HARDWARE	101	CPU Load high.

The following resources are identified.

Resource id	Resource name
1.1.1.50.1	CPU

The following nodes are found in the system.

Node	Node name
10.1.1.1	Node A
10.1.1.2	Node B
10.1.1.3	Node C
10.1.1.4	Node D
10.1.1.5	Node E

When the CPU load is high in nodes A, C and D the AAL in the ESA will hold the following alarms.

```
HARDWARE:101:1.1.1.50.1 from 10.1.1.1 (Node A)
HARDWARE:101:1.1.1.50.1 from 10.1.1.3 (Node C)
HARDWARE:101:1.1.1.50.1 from 10.1.1.4 (Node D)
```

The same alarm is triggered three times, but since the sources are different the AAL will hold three alarms.

Let's assume that CPU load on node C is lowered. The alarm is cleared. This will lead to the following AAL content.

```
HARDWARE:101:1.1.1.50.1 from 10.1.1.1 (Node A)
HARDWARE:101:1.1.1.50.1 from 10.1.1.4 (Node D)
```

3.2.4.2 Alarm Clear Control

The Alarm Clear Control (ACC) feature prevents sending misleading alarm messages to the OSS. It is not possible to clear an alarm that is not active. The ACC controls the flow of *alarm clear* messages.

If an alarm clear is sent when there is no active *alarm raise* corresponding to the alarm clear sent and the ACC feature is active, the alarm clear is dropped and **not** sent to the OSS. If the ACC is inactive the alarm clear message is sent to the OSS as is.

3.2.4.3 Alarm Flooding Control

The Alarm Flooding Control (AFC) feature prevents sending repeated alarm messages to the OSS. It is not recommended to repeat sending of alarms that are already active. The AFC controls the flow of *alarm raise* messages.

If an alarm raise is sent when there are already corresponding active *alarm raise* messages in the AAL and the AFC feature is active, the alarm raise is dropped and **not** sent to the OSS. If the AFC is inactive in this situation, the time stamp is updated only.

The alarm attributes that are used to identify flooding in the active alarm in the AAL are the same as the ones listed in chapter Section 3.2.4.1 on page 7.

3.2.5 Send Alarm

The send *alarm* operation is about triggering alarms in the ESA. Normally this means that a SNMP message is being sent to the OSS. This will happen depending on the configuration setup.

The SNMP message sent referring to *alarm* is either “alarm raise” or “alarm clear”. The alarm is stateful, which means it is active until it is cleared. The alarm raise is sent to indicate an error has occurred and the error is active until the error is resolved and an alarm clear is sent.

The SNMP messages sent as alarm raise and alarm clear respectively are defined in MIB named ERICSSON-SNF-ALARM-MIB.

All alarms sent to the OSS are logged.

3.2.6 Send Event

The send *event* operation is about triggering events in the ESA. Normally this means that a SNMP message is being sent to the OSS. This will happen depending on the configuration setup.

The SNMP message sent referring to *event* is a “stateless alarm”, which means it is sent as an alarm, but **not** stored in the AAL as an active alarm. An event can not be cleared.

Please note that the events have severities like the alarms and thus they do indicate errors. Error indications should to as large extent as possible be indicated with alarms. Events should *only* be used when it is impossible to find the clear trigger that triggers the alarm clear message.



The SNMP message sent as event is defined in MIB named ERICSSON-SNF-EVENT-MIB.

All events sent to the OSS are logged.

3.2.7 Active Alarm List

The Active Alarm List (AAL) is a list of active alarms. The AAL is useful for the OSS when it wants to issue an alarm synchronization procedure in order to verify that the alarm status for a node is the same in the agent as in the OSS itself.

Also, the AAL is of course useful for the local technician at the node to understand what errors are and have been active. The AAL combined with the logs will help finding the root cause of the problem(s).

The content of the AAL is Active Alarms, which are described in Section 3.2.4 on page 7.

3.2.8 Alarm Cleared List

The Alarm Cleared List is a list of cleared alarms. It can be seen as a history log of system faults being solved.

Please note that this list, compared to the Active Alarm List, is limited in size. Default size is set to show the last 100 cleared alarms. The size is configurable though. To modify the length of this table, see Reference [2].

3.2.9 Event History List

The Event History List is a list of sent events. It can be seen as history log of events that have been sent from the system.

Please note that this list, compared to the Active Alarm List, is limited in size. Default size is set to show the last 100 sent events. The size is configurable though. To modify the length of this table, see Reference [2].

3.3 Configuration Overview

The alarm definition configuration XML file(s) is used for configuration of the FMA Alarm Service feature. See Figure 1.

The following characteristics apply to the configuration file(s).

The alarm definition configuration XML files:

- are built up using the XML format where content and format are steered by an XML Schema.

See Section 3.4 on page 12.

- are put into a specific alarm definition configuration directory.

See Section 3.5 on page 16.

- are loaded at ESA startup and at file modification detection during runtime.

See Section 3.6 on page 17.

Please note that the old ESA alarm definition format, that is the CSV format used in ESA 1.0, is not possible to use in ESA 16. Also the ESA 2.0 and 3.0 formats are not fully backward compatible with the ESA 16 format even though it is in XML format, and look very similar. The ESA alarm definition format used in ESA 4 is however fully compatible with the ESA 16. To use the content of the older configuration files than ESA 4, they must be converted to the ESA 16 format.

3.4 Configuration Format

3.4.1 Format

The following is an example of an alarm definition:

```
<alarmDefinitions>
  <alarmSpecification active="yes">
    <moduleId>MYMODULE</moduleId>
    <errorCode>12</errorCode>
    <severity>3</severity>
    <modelDescription>A fault occurred!</modelDescription>
    <activeDescription>Process fault!</activeDescription>
    <eventType>1</eventType>
    <probableCause>1024</probableCause>
    <documentation>
      ...
    </documentation>
  </alarmSpecification>
</alarmDefinitions>
```

The above format shows all alarm data that are used in the actual alarm message sent from the ESA.

The following part shows the documentation section of the alarm definition. It is optional, but highly recommended, to use. Having the alarm documentation in relation to the alarm definition makes it useful for the local technician to easily read alarm documentation in the XML configuration file, but it can also be used to generate alarm lists and alarm documentation content for different formats, such as Web based and PDF formats.



The following is an example of an alarm documentation section in an alarm definition:

```
<documentation>
  <description>
    The process XXX failed to process message YYY.
  </description>
  <alarmingObject>
    Component ABC.
  </alarmingObject>
  <raisedBy>
    A process could not complete its tasks since it could not
    access the database.
  </raisedBy>
  <clearedBy>
    The process self control functions will try again and, if
    successful, clear the alarm.
  </clearedBy>
  <proposedRepairAction>
    Check log ZZZ and find detailed error indication. Restart
    process XXX.
  </proposedRepairAction>
</documentation>
```

See Section 3.7 on page 17 for examples of fully working alarm configuration files.

3.4.2 Mandatory Parameters

3.4.2.1 XML Attribute: **active**

This attribute gives the possibility to suppress alarms that are of no interest. For example an OSS technician could find a low risk alarm being sent too often having no critical meaning. Instead of removing the entire alarm configuration from the configuration file, the attribute `active` is simply set to `no`.

By using the attribute in the configuration and setting it to `no` the alarm is suppressed, which means it is not sent outside the ESA. In other words; the OSS will not receive it and the OSS can not synchronize it.

However, the application holding the triggering mechanism that triggered the alarm will continue to work as normal, having in mind that an alarm was sent and will also clear the alarm if setup to do so. The application is therefore not affected.

If the attribute is changed in a live environment, keep in mind that changing this attribute on an alarm that is already in the AAL, will lead to that the alarm is not possible to clear. By using a complete restart the AAL is cleared and the situation does not occur.

Since the alarm is suppressed and not visible in the AAL nor the logs it is not possible to trace the triggering of the alarm when the attribute is set to `no`. The tracing is possible only when the ESA is running in log level debug.

3.4.2.2 XML Element: `moduleId`

The `moduleId` is an elective text string used to group the alarms. Do not leave it empty. The combination of this attribute and the attribute `errorCode` must be unique, because together they constitute the Alarm Identity.

The module identity can have maximum 32 characters and only letters (a-z and A-Z) and digits (0-9) are allowed. Character space is not allowed.

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 receiving the alarms to understand the alarm purpose and content*. The name should not be chosen only to simplify design and implementation.

When it comes to creating the configuration file it is recommended to have all the alarms within one module within one configuration file. The alarm definition configuration files will be kept smaller and easier to read and maintain.

3.4.2.3 XML Element: `errorCode`

The `errorCode` is a number, greater than zero, which is used to give the alarm a unique identity. The combination of the attribute `moduleId` and this attribute must be unique, because together they constitute the Alarm Identity.

The error code is defined as a 32 bit value. Please keep in mind that error code should be chosen *to simplify for the technicians that are receiving the alarms to understand the alarm purpose and content*. The error code should not be chosen only to simplify design and implementation. Therefore, using an error code with more than six digits is not recommended.

When an alarm is undefined it is heavily recommended to not reuse the error code of the removed alarm. Systems running in live environment are configured with the error codes in use. If the system is upgraded and the error codes known from before suddenly have different meanings, the technicians monitoring the system may be confused.

3.4.2.4 XML Element: `severity`

The `severity` is the severity of the alarm that is either of the following.

- ☐ (1) Cleared/Normal (do not use)
- ☐ (2) Indeterminate (do not use)
- ☐ (3) Critical



- ☐ (4) Major
- ☐ (5) Minor
- ☐ (6) Warning

Please note that the severity 1 shall not be used in the ESA alarm definition configuration files. The clearing of an alarm is done by triggering an alarm clear message and not by using the severity 1.

Also, the severity 2 shall be avoided, if possible. When defining alarms the severity should in fact be known and not by configuration set to indeterminate. That is misleading to the user of the alarm information.

3.4.2.5 XML Element: **modelDescription**

This attribute provides a textual description of the active alarm and is also known as *alarm slogan* or *alarm title*. This is the alarm string suitable to display to operators. This attribute takes a free text of any format.

The maximum number of characters allowed is 255.

3.4.2.6 XML Element: **activeDescription**

This attribute provides a textual description of the active alarm. While the `modelDescription` is more of a slogan, this text is a more detailed description of the fault that has occurred.

This text is generated at runtime at the time of triggering the alarm and should provide more real time data allowing the operator to locate the resource for which the alarm is being generated.

This information is not intended for automated tools. This attribute takes a free text of any format.

Please note that the string defined in the configuration file is used in case the string is not dynamically created at trigger time.

The maximum number of characters allowed is 255.

3.4.2.7 XML Element: **eventType**

The *Event Type* of the alarm, as specified by the ITU. Valid values are stated in the MIB **ITU-IANA-ALARM-TC**. Also, see Section 8.2 on page 51.

3.4.2.8 XML Element: **probableCause**

The *Probable Cause* of the alarm, as specified by the ITU. Valid values are stated in the MIB **ITU-IANA-ALARM-TC**. Also, see Section 8.3 on page 52

3.4.3 Optional Parameters

3.4.3.1 XML Element: documentation

This element contains sub elements that describes the alarm, why the alarm was raised and proposed actions. The data entered within this element is pure documentation and has no effect on the ESA processes.

3.4.3.2 XML Element: description

Give a high level description of the alarm.

Example:

The process XXX failed to process message YYY.

3.4.3.3 XML Element: alarmingObject

Describe which component, function or service that triggered the alarm.

Example:

Component ABC.

3.4.3.4 XML Element: raisedBy

Describe the cause for raising the alarm.

Example:

A process could not complete its tasks since it could not access the database.

3.4.3.5 XML Element: clearedBy

Describe what will clear the alarm. If the alarm is not cleared manually it is recommended to enter "*The alarm must be manually cleared.*".

Example:

The process self control functions will try again and, if successful, clear the alarm.

3.4.3.6 XML Element: proposedRepairAction

Give a solution or at least a proposal for trouble shooting and describe how to resolve the alarm situation.

Example:

Check log ZZZ and find detailed error indication(s).
Solve the issue and/or restart process XXX.



3.5 Configuration Deployment

The directory path to use for the alarm definition configuration files is configurable.

See configuration parameter `fmAlarmDefinitions` in configuration file `mainCfg.xml` described in Reference [2].

The default alarm definition configuration directory is:

□ `{esa basedir}/conf/fmAlarmDefinitions/`

3.6 Configuration Activation

The alarm definitions are loaded at the startup of ESA. The ESA loads all the XML files found in the specified alarm definition configuration directory.

The alarms that are configured to be *inactive* are also loaded and can be triggered, but are not visible in the AAL nor sent to the OSS.

The ESA supports reloading of configuration files during runtime, which means it will detect changes in the alarm definition configuration files at runtime. If any file is added, removed and/or modified the ESA will load all the XML files, verify the content and, if the validation is successful, use the new alarm definitions. This means that there is no need to restart the ESA to activate changes made to the alarm definition configuration.

Error handling during load/reload of configuration files:

- **Startup**

If any configuration file is found erroneous during startup of ESA the error is logged and the ESA does not start.

- **Runtime Reload**

If any configuration file is found erroneous during runtime reloading the ESA will simply disregard the configuration files on disk, will not load any of them and keep running the ones being loaded at the last successful load/reload.

3.7 Examples

The following is an example of an alarm definition configuration file containing only **one** alarm definition.

```
<alarmDefinitions>
  <alarmSpecification active="yes">
    <moduleId>TEST</moduleId>
    <errorCode>47</errorCode>
    <severity>3</severity>
  </alarmSpecification>
</alarmDefinitions>
```



```
<modelDescription>Test alarm.</modelDescription>
<activeDescription>
  This is a test alarm with the severity Critical.
</activeDescription>
<eventType>11</eventType>
<probableCause>22</probableCause>
</alarmSpecification>
</alarmDefinitions>
```

The following file contains a number of alarm definition examples:

```
{esa basedir}/conf/examples/example_fmalarmdefinition.xml
```



4 Alarm Translation

4.1 Introduction

The alarm translation is about setting up *translation* rules, which is also known as *mapping* rules, for the ESA, when handling SNMP alarms in other formats than the ESA format. The translation provides the benefit of having one single format of all the SNMP alarms towards the OSS, from any number of sources in any format. Also, the translation comes with the possibility to filter out and simply suppress alarms that are not to be sent further. The translation rules are defined in one or several XML files, which are read and loaded by the FM Agent.

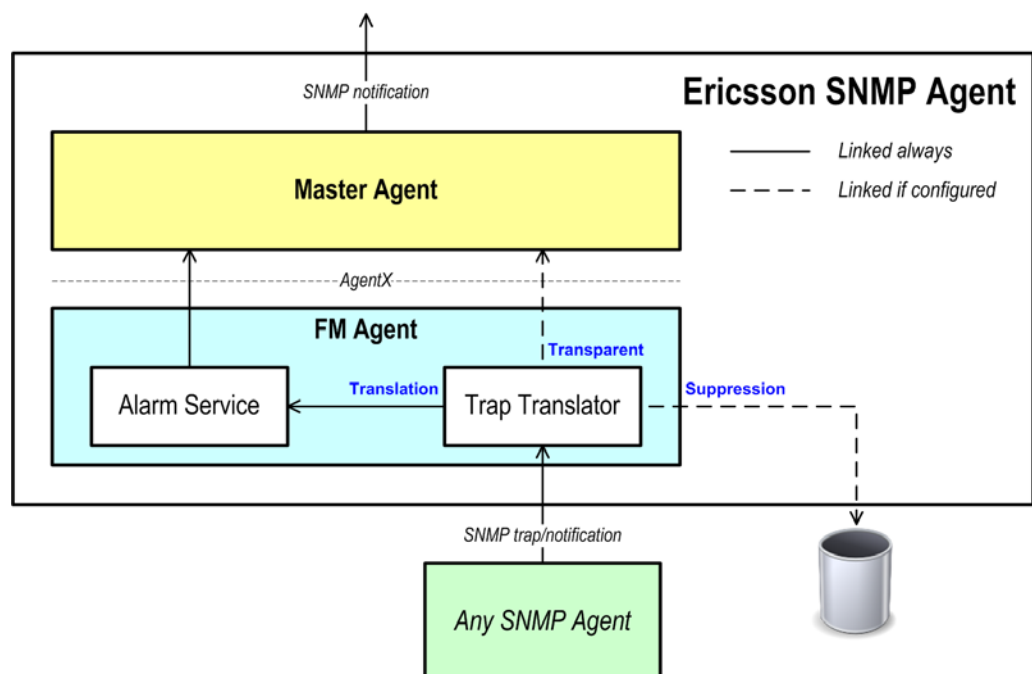


Figure 2 The alarm flows using the alarm translation rules in ESA Trap Translator.

There are three translation functions, “translation”, “suppression”, and “transparent”, available in the ESA FMA. They are described in more detail in the following sub sections.

The logic is as follows.

- The first matching “suppress” statement, in any of the XML configuration files, results in that the trap is suppressed, which means it is dropped. Nothing else happens.
- Each matching “translation” statement, one or several depending on the configuration, results in that the trap is handled as defined in the “output”

statement. The three possibilities are “raise”, “clear”, and “event”, which results in that an alarm/event is sent to the OSS.

- If there is a matching “transparent” statement the trap is sent untouched through the ESA, which means it is sent to the OSS in the format used by the sending agent.
- If the trap does not match any statement, which means that no “translation”, “suppression”, or “transparent” is triggered, it is logged to file and is considered being “not translated”. An analysis of the not translated trap(s) in the log file should be done. Maybe there are alarms that are of interest that needs translation configurations to be setup in order to get the ESA to handle them properly.

4.2 Function Description

4.2.1 Alarm Reception

The FMA Trap Translation feature in the ESA supports receiving traps in any SNMP format, which means SNMP v1, v2c and v3. The FMA can receive traps at any time as long as it is running.

The subagent that sends traps to the ESA must be configured to send its traps to the FM Agent SNMP port. See Reference [2] for further information about what ports to use for integrating SNMP agents with the ESA.

Please note that setting up alarm reception for SNMPv3 traps is done by configuring USM as described in Reference [2].

4.2.2 Condition

Any agent can send traps to the FMA. The FMA is not looking at the MIB itself, but the integrator mapping the incoming alarm to an ESA alarm must use the MIB to identify the alarm content and the different varbinds in the trap to integrate.

To create a match, one or several *conditions* are setup. A condition is setup by identifying which varbind to look at, identify a matching data type, oid and/or the value.

The *index* is mandatory in the condition. The index indicates the varbind number in the trap from 1 to n , where n is the maximum number of varbinds in the trap received. ESA will verify that the incoming trap has the varbind specified. If index 7 is specified and the received trap only have 5 varbinds, there is no match.

The *data type* is optional. The data type indicates the data type to match in the varbind index being matched in the trap received. If the trap received does not have the data type specified in the index specified, there is no match.



The *oid* is optional. The oid indicates the varbind oid to match for the varbind index being matched in the trap received. If the trap received does not have the oid specified in the index specified, there is no match. This parameter allows the use of regular expressions.

The *value* is optional. The value indicates the varbind value to match for the varbind index being matched in the trap received. If the trap received does not have the value specified in the index specified, there is not match. This parameters allows the use of regular expressions.

The condition statement is valid for all operations, meaning translation, suppression and transparent. The succeeding action when a match is found is however different for the three operations.

4.2.3 Output

The output is a specification of which message to trigger. The supported messages are alarm and event. See Section 3 on page 5, which describes the alarm definitions.

The specification of an output matches the alarm definitions, which means using *module* and *error code* to point at a defined alarm to trigger. The *resource identity* is used to identify the alarming resource. The *text* provides a space to add additional information about the alarm triggered. Finally, the *originating source IP* can be used to specify to which node the alarm actually belongs to in case the ESA is not located on the node that is running the agent sending traps to the ESA.

4.2.4 Suppress

Suppression is about filtering out SNMP messages that are not of interest.

Some of the traps received might not be of interest for the OSS technicians. The FMA suppress feature can be used to suppress the traps. This means that the trap will be dropped and not handled any further.

The following is a short description of the suppress procedure.

- The trap/notification is received by the FMA.
- The FMA configuration is used to find out if there is a *suppress* match for the received alarm.
- A match is found, meaning the trap/notification is dropped.

Referring to the names of elements and attributes in Section 4.4 on page 23, the suppression works as follows; When a trap is received and matches the IP address(es) in the element *senderIp*, its value pointed out by the element *index* is matched against the value in the element *value* of the data

type found in the element `type` and/or the varbind oid is matched against the element `oid`. If there is a match, the trap is suppressed.

4.2.5 Translation

Translation is about translating (or transforming) SNMP messages from any SNMP format to the Ericsson ESA format.

An OSS might have to be able to support several different SNMP alarm formats, which can cause integration problems. The FMA translation feature makes it possible to convert SNMP traps/notifications received in any SNMP format into SNMP traps/notifications in the Ericsson SNF format, which is the format used by the ESA. This way, a convergent single format from any alarming source is facilitated. This leads to a single integration to the OSS and there is only one alarm format to handle and understand. Additional nodes added to the network using the ESA means the OSS integration is very simple.

The following is a short description of the translation procedure.

- The trap/notification is received by the FMA.
- The FMA configuration is used to find out if there is a *translation* match for the received alarm.
- A match is found, meaning an ESA alarm or event is triggered.
- The SNMP alarm in ESA format is sent to the OSS.
- If an alarm is sent (not event) the alarm also ends up in the ESA AAL.

Referring to the names of elements and attributes in Section 4.4 on page 23, the translation works as follows. When a trap is received and matches the IP address(es) in the element `senderIp`, its value pointed out by the element `index` is matched against the value in the element `value` of the data type found in the element `type` and/or the varbind oid is matched against the element `oid`. If there is a match, the trap defined in the element `output` is triggered. The attribute `type` indicates the alarm type to trigger, which means `raise`, `clear` or `event`. The trap is converted to an alarm identified with elements `moduleId` and `errorCode` holding data about `resourceId`, `text` and `origSourceIp`. The alarm is finally sent to the OSS, eventually processed in the AAL, and logged.

4.2.6 Transparent

Transparent is about handling SNMP messages from any SNMP agent transparently through the ESA.

There might be cases when the OSS wants the original traps instead of transforming them to the ESA format. The FMA translation feature makes it



possible to define a selection of traps from the integrated agent that are to be handled transparently.

The following is a short description of the translation procedure.

- The trap/notification is received by the FMA.
- The FMA TT configuration is used to find out if there is a *transparent* match for the received alarm.
- A match is found.
- The SNMP alarm is sent to the OSS in the original format.

A trap matching the conditions, and having the element `transparent`, will be passed through the ESA, untouched. Refer to the names of elements and attributes in Section 4.4 on page 23.

Note: Notice that the source IP address will **always** show the IP address of the ESA, which means it will be changed from the originating SNMP agent IP address in case it is not the same as the ESA IP address. This means that the receiving OSS will not get the information about the IP address of the originating SNMP agent.

4.3 Configuration Overview

The alarm translation configuration XML file(s) is used for configuration of the FMA Trap Translator feature.

The following characteristics apply to the configuration file(s).

The alarm translation configuration XML files:

- are built up using the XML format where content and format are steered by an XML Schema.

See Section 4.4 on page 23.

- are put into a specific alarm translation configuration directory.

See Section 4.5 on page 33.

- are loaded at ESA startup and at file modification detection during runtime.

See Section 4.6 on page 34.



4.4 Configuration Format

4.4.1 Format

The alarm translation configuration XML file is built according to XML Schema `fmAlarmTranslation.xsd`, which comes with the ESA installation and is found in the following directory.

□ `{esa basedir}/conf/fmAlarmTranslations/`

The following is an example alarm translation:

```
<alarmTranslations>
  <trap oid=".1.3.6.1.4.1.1031.1.0.*">
    <grpSenderIP>
      <senderIp>19.72.9.21</senderIp>
      <senderIp>19.72.9.22</senderIp>
    </grpSenderIP>
    <translation>
      <condition>
        <index>2</index>
        <type>integer</type>
        <value>15</value>
      </condition>
      <output type="raise">
        <moduleId>Hardware</moduleId>
        <errorCode>1</errorCode>
        <resourceId>.1.1.9.7</resourceId>
        <text>Disk A needs replacement.</text>
      </output>
    </translation>
  </trap>
</alarmTranslations>
```

More examples are found in Section 4.7 on page 34.

4.4.2 Parameters

4.4.2.1 XML Element: alarmTranslations

The root element `alarmTranslations` consists of one or more `trap` elements, and is unique in the alarm translation file. Several `alarmTranslations` are recommended when there are multiple agents being integrated with the ESA. One XML file is recommended for each agent integrated. Also, very large XML files can be split into several smaller ones in order to ease the reading and understanding.



4.4.2.2 XML Element: trap

The element `trap` consists of the required attribute `oid`, which is the OID of the trap(s) that is to be translated. Regular expressions can be used on the OID. The trap oid is the first matching condition.

This element can hold any number of the element `senderIp`, but also one or several `translation`, `suppress` and `transparent` statements.

4.4.2.3 XML Element: senderIp

The element `senderIp` indicates the IP address of the sending network element, which is the IP address of the network element that sent the trap. If alarms are received from other IP addresses than the ones listed in the translation configuration, there is no match and no further processing is done. The element `grpSenderIP` is only a grouping element for the `senderIp` elements.

This attribute is optional. If the attribute is not defined, the trap translation configuration is valid for all IP addresses.

4.4.2.4 XML Element: suppress

The element `suppress` holds no or several `condition` elements. When the content of an incoming trap matches all the specified conditions, the trap is dropped.

Refer to Section 4.2.4 on page 21 for more information about the *suppress* procedure.

4.4.2.5 XML Element: translation

The element `translation` holds no or several `condition` elements. Also, one or several `output` elements can be defined.

When the content of an incoming trap matches all of the conditions stated for that trap, the translation results in triggering the output.

Refer to Section 4.2.5 on page 22 for more information about the *translation* procedure.

4.4.2.6 XML Element: transparent

The element `transparent` holds no or several `condition` elements. When the content of an incoming trap matches all the specified conditions, the trap is sent untouched through the ESA, which means the original SNMP format is used instead of the ESA format. Please note that the sender IP might be changed. When the sending agent is running on a different IP address than the ESA, the trap sent through the ESA as transparent means that the trap will get the ESA IP address as the sender IP address.

Refer to Section 4.2.6 on page 22 for more information about the *transparent* procedure.

4.4.2.7 XML Element: **condition**

The element `condition` holds one each of the elements `index`, `type`, `oid` and `value` where several of them are optional. Within this element, the triggering conditions are defined. Any trap matching the given combination of all the conditions specified will trigger the specified action.

- `index`

The element indicates which varbind in the incoming trap to match.

- `type` (optional)

The element specifies the data type to match. The possible data types are:

- `octetstring`
- `snmpadminstring`
- `objectidentifier` | `oid`
- `integer`
- `integer32`
- `unsigned32`
- `gauge32`
- `counter32`
- `counter64`
- `timeticks`
- `ipaddress`

- `oid` (optional)

The element holds the varbind oid to match. Regular expressions are allowed, which means bits and pieces of the varbind oid can be matched if needed.

- `value` (optional)

The element holds the value to match in the trap. Regular expressions are allowed, which means bits and pieces of the varbind value can be matched if needed.



4.4.2.8 XML Element: output

The following elements build up the output of the alarm translation.

The element `output` consists of the required attribute `type` that holds either of the following values:

- `raise`

An alarm raise is sent.

- `clear`

An alarm clear is sent.

- `event`

An event is sent.

For all selections in attribute `type` the element `output` also holds the following elements:

- `moduleId` (mandatory)

The element is a string, indicating the alarm module of the trap to be sent. See Section 3.4 on page 12 for more information about this element.

- `errorCode` (mandatory)

The element is an integer, indicating the Error Code of the trap to be sent. See Section 3.4 on page 12 for more information about this element.

- `resourceId` (mandatory)

The element contains the resource identity of the alarming source taken from the system topology

Note: Please note that `resourceId` can have a maximum length of 128 sub-identifiers.

- `text` (optional)

The element is a string, containing additional information of the trap to be translated. This element should be given as a text string, but may be left empty. Please note that the length is limited to 255 characters. If a string is given or generated that is longer than 255 characters, the string is truncated to 255 characters.

- `origSourceIp` (optional)

The element is an IP address identifying the originating source IP address, in case it is different from the one used by the ESA itself.

4.4.3 Trap Properties Reuse

4.4.3.1 Function

All incoming alarms come with properties that can be reused in the conditions and in the output. Varbind content is described in the following chapters. This chapter is about reusing trap properties and behavior using static variables.

4.4.3.2 Format

The following static variables can be used for reusing trap data from incoming traps.

@ip@ This is the source IP address, which means the IP address of the server from where the alarm originates.

@trapoid@ This is the trap object identity.

In general the variables are by the ESA replaced as is, meaning the complete IP address is used as a string. The following requires special attention as a conversion is applied when using the variables as part of the resource identity.

- If the IP is an IPv4 address equal to
`123.45.67.89`

and resource identity is specified as
`1.1.1.@ip@`

the resulting resource identity will be
`1.1.1.123.45.67.89.`

The IP address is simply reused as is since the IPv4 notation fits into the OID format.

- If the IP is an IPv6 address equal to
`2001:0db8:85a3:0042:1000:8a2e:0370:7334`

and resource identity is specified as
`1.1.1.@ip@`

the resulting resource identity will be
`1.1.1.8193.3512.34211.66.4096.35374.880.29492.`

The hexadecimal values are converted to decimal values in order to fit into a valid OID format.

Please note that an IPv6 address with leading zeroes, such as `::0001`, which in fact means `0000:0000:0000:0000:0000:0000:0000:0001`, will end up with resource identity `1.1.1.0.0.0.0.0.0.0.1.`



4.4.3.3 Example

Assume a SNMP alarm with trap OID "1.3.6.1.4.1.1977.0.34" holding the following varbinds is received in the Trap Translator from IP address 123.45.67.89.

```
1: INTEGER: 234                (Error Code)
2: STRING: Device down        (Error Message)
3: INTEGER: 15                (Device Type)
4: STRING: Disk A             (Device Name)
5: INTEGER: 1                 (Event Type)
6: INTEGER: 1024              (Probable Cause)
```

The following example gives an introduction to reuse of trap properties. Please focus on the bold items.

```
<trap oid="1.3.6.1.4.1.1977.0.34">
  <translation>
    <condition>
      ...
    </condition>
    <output type="raise">
      <moduleId>DEVICE</moduleId>
      <errorCode>234</errorCode>
      <resourceId>.1.1.9.7.@trapoid@</resourceId>
      <text>Device %4 in system @ip@ is faulty.</text>
      <origSourceIp>@ip@</origSourceIp>
    </output>
  </translation>
</trap>
```

The alarm sent from the ESA will have the following content:

```
<resourceId>.1.1.9.7.1.3.6.1.4.1.1977.0.34</resourceId>
<text>Device Disk A in system 123.45.67.89 is faulty.</text>
<origSourceIp>123.45.67.89</origSourceIp>
```

4.4.4 Trap Varbind Reuse

4.4.4.1 Function

All varbinds in the incoming alarm can be reused in the output. Often there are incoming alarms that have dynamic content that is taken from the point and time of failure. For example an alarm saying "CPU load high" is not informative compared to "CPU raised above 95%, currently at 98%". The following chapters describes the reuse capabilities of incoming varbinds.



4.4.4.2 Format

Assume an SNMP alarm with the following varbinds is received in the Trap Translator.

1: INTEGER: 234	(Error Code)
2: STRING: Device down	(Error Message)
3: INTEGER: 15	(Device Type)
4: STRING: Disk A	(Device Name)
5: INTEGER: 1	(Event Type)
6: INTEGER: 1024	(Probable Cause)

The following example gives an introduction to varbind reusability. Please focus on the bold items.

```
<translation>
  <condition>
    ...
  </condition>
  <output type="raise">
    <moduleId>DEVICE</moduleId>
    <errorCode>234</errorCode>
    <resourceId>.1.1.9.7.%3</resourceId>
    <text>Device %4 of type %3 is faulty.</text>
  </output>
</translation>
```

The alarm sent from the ESA will have the following content:

```
<resourceId>.1.1.9.7.15</resourceId>
<text>Device Disk A of type 15 is faulty.</text>
```

4.4.5 Trap Data using Regular Expressions

4.4.5.1 Function

All elements and attributes can be matched against Java regular expressions for pattern purposes. Often there are incoming alarms that have an unknown or too dynamic content to create a good matching condition, but with regular expressions the possibility to create smarter and more dynamic configurations is given.

4.4.5.2 Format

The following example gives an introduction to what is possible. Please focus on the bold items.

```
<translation>
  <condition>
    <index>1</index>
    <type>snmpadminstring</type>
    <value>(.*)</value>
```



```

</condition>
<condition>
  <index>2</index>
  <type>snmpadminstring</type>
  <value>.*error in (.* ) message.*</value>
</condition>
<output type="raise">
  <moduleId>GRPMSG[2:1]</moduleId>
  <errorCode>123 [1:1]</errorCode>
  <resourceId>.1.1.9.7. [2:1]</resourceId>
  <text>Error in message type: [2:1]</text>
  <origSourceIp>10.1.100. [1:1]</origSourceIp>
</output>
</translation>

```

In the example above the strings [1:1] and [2:1] are used in the output elements. This kind of dynamic setup on all output elements are not mandatory and probably not seen very often. It's used in all places here only to show the capability and the different output handling for the different output elements, which is described in succeeding chapters.

4.4.5.3 Condition

First, let's focus on the regular expression (.*).

The use of regular expressions in the `condition` statements with surrounding parenthesis means that the ESA saves the outcome of the regular expressions. The outcome can therefore be used later, which means in the `Output` statements. If regular expressions are used *without* surrounding parenthesis, the outcome is not saved.

The format of the regular expressions to use is the one described in `java.util.regex`.

4.4.5.4 Match

Assume the ESA Trap Translator receives an alarm that looks like the following:

Varbind	Content
1	88
2	There is an error in ABC message process.
3	minor
4	...more alarm data...

The second condition in the example configuration above will lead to a successful match since index 2, which also means varbind 2, in the trap contains a string that matches the condition. There are three regular expressions in the condition statement `value`. In the beginning there is one, the ".*", at the end another one, the ".*" and in the middle a third one, the

"(.*)". Only the middle one is stored by the ESA, by using surrounding parenthesis, and is therefore considered the first one, *number 1*.

In the output configuration the statement `[2:1]` is used. The value 2 simply refer to condition index 2 and the value 1 is the number of the regular expression on index 2. In other words, the statement `[2:1]` will contain string `ABC`, which is captured from string `There is an error in ABC message process`.

The first condition is managed in the same way, which results in statement `[1:1]` holding string `88`.

We now have two matches and we have used the result from the regular expression in the output specification. What does the output look like? The output looks differently for different output elements.

4.4.5.5 Output - Module

The regular expression that read `ABC` from the incoming alarm is concatenated with the string according to the output element `moduleId`.

In the example from above the match `ABC` in the output `GRPMSG[2:1]` will look like the following:

Format:
`GRPMSG[2:1]`

Result:
`GRPMSGABC`

4.4.5.6 Output - Error Code

The regular expression that read `88` from the incoming alarm is concatenated with the string according to the output element `errorCode`.

In the example from above the match `88` in the output `123[1:1]` will look like the following:

Format:
`123[1:1]`

Result:
`12388`

4.4.5.7 Output - Resource Id

The regular expression that read `ABC` from the incoming alarm is concatenated with the string according to the output element `resourceId` with a character by character ASCII representation of the matching string.



In the example from above the match ABC in the output `.1.1.9.7.[2:1]` will look like the following:

Format:

`.1.1.9.7.<num of chars>.<ascii char 1>.<ascii char 2>...\<ascii char n>`

Result:

"ABC" contains 3 characters with ASCII 65, 66 and 67.
= `".1.1.9.7.3.65.66.67"`

Keep in mind that if the captured match is a string "45", it is still considered a string and not integer, which means `"2.53.54"` is used representing "2" characters, "53" as ASCII for character 4 and "54" as ASCII for character 5.

4.4.5.8

Output - Text

The regular expression that read ABC from the incoming alarm is used within the string according to the output element `text`.

In the examples from above the match ABC in the output `Error in message type: [2:1]` will look like the following:

Format:

`Error in message type: <matching string>`

Result:

`Error in message type: ABC`

4.4.5.9

Output - Originating Source IP

The regular expression that read 88 from the incoming alarm is concatenated with the string according to the output element `origSourceIp`.

In the example from above the match 88 in the output `10.1.100.[1:1]` will look like the following:

Format:

`10.1.100.[1:1]`

Result:

`10.1.100.88`

4.5

Configuration Deployment

The directory path to use for alarm translation configuration files is configurable.

See configuration parameter `fmAlarmTranslations` in configuration file `mainCfg.xml` described in Reference [2].



The default alarm translation configuration directory is:

□ `{esa basedir}/conf/fmAlarmTranslations/`

4.6 Configuration Activation

The alarm translation configuration files are loaded at the startup of ESA.

The ESA also supports reloading of alarm translation configuration files during runtime. In case a file is added, modified or deleted, the ESA will notice and reload all the configurations without a restart.

4.7 Examples

4.7.1 Introduction

The examples in the following chapters are kept very simple in order to make them easy to follow. Normally, in a live environment, there will be many translation, suppression and transparent statements in a single XML configuration file.

The following file, which come with the ESA installation, contains a number of alarm translation examples:

`{esa basedir}/conf/examples/example_fmalarmtranslation.xml`

4.7.2 Example: Translation

The following example shows a *translation* configuration.

```
<alarmTranslations>
  <trap oid=".1.3.6.1.4.1.1031.1.0.*">
    <grpSenderIP>
      <senderIp>136.225.101.140</senderIp>
      <senderIp>136.225.101.141</senderIp>
      <senderIp>136.225.101.142</senderIp>
    </grpSenderIP>
    <translation>
      <condition>
        <index>2</index>
        <type>snmpadminstring</type>
        <value>
          .*License_TrafficRegul.(.*).CapacityUsage.*
        </value>
      </condition>
      <output type="raise">
        <moduleId>Licence_TrafficRegul</moduleId>
      </output>
    </translation>
  </trap>
</alarmTranslations>
```



```

        <errorCode>123</errorCode>
        <resourceId>.1.1.9.7</resourceId>
        <text>Alarm text: [2:1]</text>
    </output>
</translation>
</trap>
</alarmTranslations>

```

From this alarm translation file, the following can be read out.

- The trap(s) that can be translated by this configuration are the one(s) matching the OID `.1.3.6.1.4.1.1031.1.*`, where “.” is a Java regular expression.
- The trap has to originate from any of the sources `136.225.101.140`, `136.225.101.141` or `136.225.101.142`, for the translation to trigger.
- A check is performed, to determine if the trap matches the conditions, that is if “varbind 2” is of “data type string” equals the resulting value of the Java regular expression `.*License_TrafficRegul.(.*).CapacityUsage.*`.
- If there is a full match, an “alarm raise” is sent to the trap destination. The alarm consists of the information about the alarming module `Licence_TrafficRegul`, the error code `123`, the resource ID `.1.1.9.7`, and the text `Alarm Text: [2:1]`.

The latter is actually replaced by the value of the first (“[2:1]”) Java regular expression parenthesis, “(.*?)”, within the condition element using the parameter with index = “2” (“[2:1]”). If, for example varbind #2 of the trap equals “The KPI is `License_TrafficRegul.PullTraffic.CapacityUsage.Value`”, the text “[2:1]” will result in “PullTraffic”.

4.7.3 Example: Suppression

The following example shows a *suppression* configuration.

```

<alarmTranslations>
  <trap oid=".1.3.6.1.4.1.1031.1.0.*">
    <grpSenderId>
      <senderIp>136.225.101.140</senderIp>
      <senderIp>136.225.101.141</senderIp>
      <senderIp>136.225.101.142</senderIp>
    </grpSenderId>
    <suppress>
      <condition>
        <index>1</index>
        <type>integer</type>
        <value>60</value>
      </condition>
    </suppress>
  </trap>
</alarmTranslations>

```

```
</trap>
</alarmTranslations>
```

From this alarm translation file, the following can be read out.

- The trap(s) that can be translated by this configuration are the one(s) matching the OID `.1.3.6.1.4.1.1031.1.*`, where “.” is a Java regular expression.
- The trap has to originate from any of the sources `136.225.101.140`, `136.225.101.141` or `136.225.101.142`, for the translation to trigger.
- A check is performed, to determine if the trap matches the conditions, that is if the “varbind 1” is of “data type integer” and equals to value `60`.
- If there is a full match, the trap is suppressed, which means it is dropped and not processed further. No alarm is sent anywhere, and nothing is logged.

4.7.4 Example: Transparent

The following example shows a *transparent* configuration.

```
<alarmTranslations>
  <trap oid=".1.3.6.1.4.1.1031.1.0.*">
    <grpSenderIP>
      <senderIp>136.225.101.140</senderIp>
      <senderIp>136.225.101.141</senderIp>
      <senderIp>136.225.101.142</senderIp>
    </grpSenderIP>
    <transparent>
      <condition>
        <index>3</index>
        <type>integer</type>
        <value>88</value>
      </condition>
    </transparent>
  </trap>
</alarmTranslations>
```

From this alarm translation file, the following can be read out.

- The trap(s) that can be translated by this configuration are the one(s) matching the OID `.1.3.6.1.4.1.1031.1.*`, where “.” is a Java regular expression.
- The trap has to originate from any of the sources `136.225.101.140`, `136.225.101.141` or `136.225.101.142`, for the translation to trigger.
- A check is performed, to determine if the trap matches the conditions, that is if “varbind 3” is of “data type integer” and equals the value `88`.



- If there is a full match, the trap is sent to the trap destination as is, which means without being translated.





5 Command Line Interface

5.1 Introduction

During runtime the FM Agent is ready to accept alarm triggers and hold active alarms and alarm data, which is made available to the OSS using SNMP. But, also the technician working locally with the system could benefit from looking at the data captured and produced by the FM Agent.

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

The following CLI commands are available for the FM Agent:

☐ `fmactivealarms`

☐ `fmsendmessage`

5.2 Command: View Active Alarms

5.2.1 Description

This command is used for viewing the active alarms that have been triggered in the FM Agent.

5.2.2 Command Format

The command format:

fmactivealarms [*<filter>*]

The command attributes are the following:

filter	The filter to use for narrowing down the command output.	
	-s <severity>	Filter alarms on Severity.
	-m <moduleid>	Filter alarms on Module Identity.
	-l <ipaddr>	Filter alarms on IP Address.



severity	Show alarms that belongs to the specified Severity only.
3	Critical
4	Major
5	Minor
6	Warning
	Add character '+' to also include the severities higher than the specified one, such as "3+" or "Critical+".
moduleid	Show alarms that belongs to the specified Module Identity only.
ipaddr	Show alarms that belongs to the specified IP Address only.

5.2.3 Command Execution

The command prints all active alarms found in the ESA AAL.

5.2.4 Command Output

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

```
# fmactivealarms
Active alarms:
```

If there are active alarms, the command output is a complete list of all active alarms. The output example below shows three active alarms.

```
# fmactivealarms
Active alarms:
!-----
Module           : DATABASE
Error Code       : 11
Resource Id      : 1.1.1.2.72
Timestamp First  : Mon May 05 09:45:03 CEST 2014
Repeated Counter : 1
Timestamp Last   : Mon May 05 09:45:03 CEST 2014
Model Description : This is fault number 1!
Active Description : The component XXX has an error.
Event Type       : 2
Probable Cause   : 119
Severity         : major
Orig Source IP   : 10.1.100.88
Sequence Number  : 1
-----!
!-----
Module           : DATABASE
Error Code       : 22
```



```

Resource Id      : 1.1.1.2.9
Timestamp First  : Mon May 05 09:46:12 CEST 2014
Repeated Counter : 3
Timestamp Last   : Mon May 05 09:50:37 CEST 2014
Model Description : This is fault number 2!
Active Description : The component YYY has an error.
Event Type       : 1
Probable Cause   : 1024
Severity         : minor
Orig Source IP   : 10.1.100.89
Sequence Number  : 2
-----!
!-----
Module           : DATABASE
Error Code       : 33
Resource Id      : 1.1.1.2.21
Timestamp First  : Mon May 05 09:47:22 CEST 2014
Repeated Counter : 1
Timestamp Last   : Mon May 05 09:47:22 CEST 2014
Model Description : This is fault number 3!
Active Description : The component ZZZ has an error.
Event Type       : 1
Probable Cause   : 1024
Severity         : critical
Orig Source IP   : 10.1.100.90
Sequence Number  : 3
-----!

```

The filter options give the same alarm output format, but show only the alarms that matches the specified alarms.

The following shows a few example filter operations.

Example 1) Show alarms with severity 'critical'.

```
# fmactivealarms -s 3
```

Example 2) Show alarms with severity 'major' and higher, which means 'major' and 'critical'.

```
# fmactivealarms -s Major+
```

Example 3) Show alarms with module identity 'HARDWARE'.

```
# fmactivealarms -m HARDWARE
```

Example 4) Show alarms that originates from IP address '19.72.9.21'.



```
# fmaactivealarms -i 19.72.9.21
```

5.3 Command: Send Message

5.3.1 Description

This command is used for triggering alarms and events.

5.3.2 Command Format

The command format:

```
fmsendmessage <action> <moduleId> <errorCode> <resourceId>  
[<activeDescr>] [<sourceIp>]
```

The command attributes are the following:

action	The type of action to trigger. <ul style="list-style-type: none">-r Trigger 'raise alarm' message.-c Trigger 'clear alarm' message.-e Trigger 'event' message.
moduleId	The module identity of the alarm to trigger.
errorCode	The error code of the alarm to trigger.
resourceId	The resource identity of the alarming object.
activeDescr	The text string to send in the alarm message. Default string: The configured string in the Alarm Definition configuration file.
sourceIp	The IP address to use as the alarm originating source. Note: The command can take hostname, IPv4 and IPv6 addresses. Default source: The IP address of the node where the command is executed.



5.3.3 Command Execution

The command creates an alarm raise, an alarm clear or an event message with alarm data based on the alarm definition configuration of the module identity and error code given. The message is sent to the defined trap destination(s).

If attribute active description is given, the text specified overrides the text specified for element `activeDescription` in the Alarm Definition configuration file.

If attribute originating source IP is given, it is used as the originating source IP even though it does not match the IP address(es) of the node where the command is executed.

Please note that the command attribute originating source IP can take both hostnames and IP addresses. If for some reason the hostname is to be sent as an active description text and only five attributes are given to the command, the ESA will first verify if the fifth parameter is a valid hostname and if not it will use it as text string for the active description. If the fifth parameter is found being a valid hostname, the ESA will use the fifth parameter as an originating source IP address instead of putting it as a string into the active description, which instead will be read from the `activeDescription` element in the Alarm Definition configuration file.

5.3.4 Command Output

If the specified alarm is successfully triggered, the command does not give any output.

If the specified alarm could not be triggered, the command returns an error message.





6 Application Programming Interface

6.1 Introduction

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

- **Alarm Definition**
See Section 6.2 on page 45.
- **Alarm**
See Section 6.3 on page 45.
- **Active Alarm List**
See Section 6.4 on page 46.
- **Subagent AAL Synchronization**
See Section 6.5 on page 46.

Be aware of that this section describes the use of APIs only. The definition of alarms, which are triggered from the APIs, is described in Section 3 on page 5 and it is therefore assumed that the reader is familiar with ESA alarm definition terminology when using the API.

6.2 Interface: Alarm Definition

This interface gives the API user the possibility to read alarm definitions configured in the ESA from Java code.

Interface: `com.ericsson.esa.fmagent.rmi.IAlarmDefinition`

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`.

6.3 Interface: Alarm

This interface gives the API user the possibility to trigger the sending of SNMP notifications from Java code. The alarms are sent as SNMP messages to all the trap destinations defined in the ESA.

Interface: `com.ericsson.esa.fmagent.rmi.IAlarm`

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`.

6.4 Interface: Active Alarm List

This interface gives the API user the possibility to work directly on the ESA AAL from Java code. The API supports AAL operations, such as reading AAL alarms, adding alarms to the AAL and removing alarms from the AAL.

Interface: `com.ericsson.esa.fmagent.rmi.IActiveAlarmList`

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`.

6.5 Interface: Subagent AAL Synchronization

6.5.1 Introduction

A system usually have one or several sources with information about the current alarm status. It could be an SNMP agent holding an AAL or basically any other data source, such as log files, that is holding information about active alarms. Using the Subagent AAL Synchronization (SAS) Java RMI API makes it possible to fill the ESA AAL with alarm information during startup of the ESA in order to keep the ESA synchronized with the active alarms in other data sources in the system.

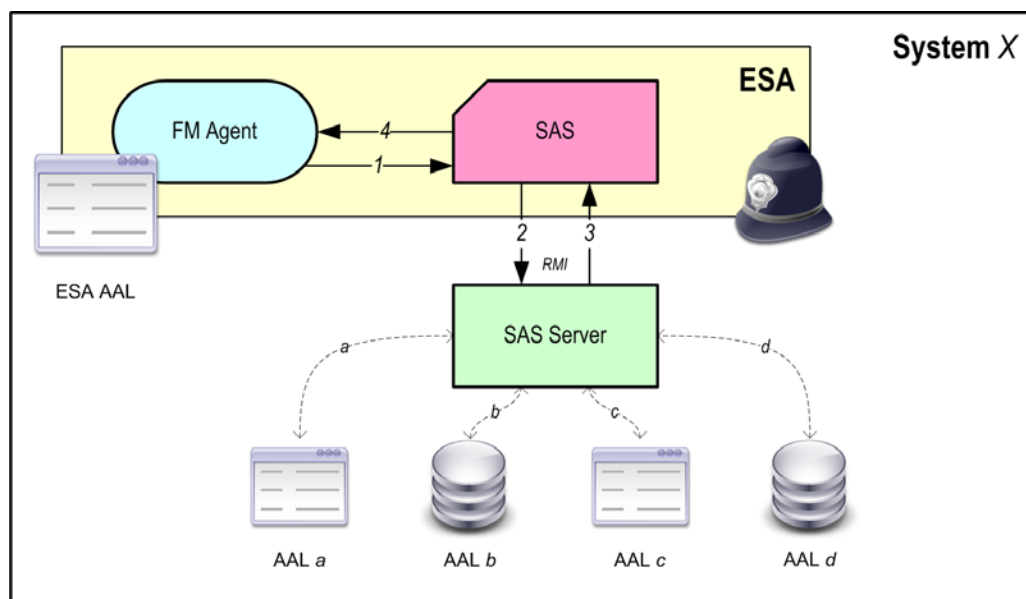


Figure 3 An overview of the ESA SAS feature.



When the ESA starts up the ESA AAL is empty. If you find it important that the ESA is loaded with correct alarm status at start up, you can create a SAS server that loads the ESA AAL with active alarms. The process starts at ESA start. If a SAS server is defined the ESA is configured to trigger the SAS server using a RMI call (1). The SAS server is triggered (2), which starts the alarm collection process. The SAS server captures alarm information from all necessary AALs (a to d) and returns the information to the ESA API (3). All the alarms are put into the ESA AAL (4) and the ESA is finally fully started.

It is important to notice that incoming alarms to the ESA, such as alarms triggered on the ESA API or alarms that come to the ESA Trap Translator, are **not** handled by the ESA during the SAS sequence. They are simply dropped. When the SAS synchronization is started the SAS server is responsible for providing the AAL to use and therefore no new alarms during that .

Also, it is important to set a valid timer for the synchronization process. If the SAS server takes too long time to respond to the ESA, the ESA stops the synchronization and starts up with an empty AAL.

6.5.2 Work Process

In order to use the SAS process the following work must be done:

1. Create a SAS RMI server used for synchronizing alarms with subagents.
2. Configure the ESA to target the SAS server at startup. See Reference [2].
3. Restart the ESA.

6.5.3 Interface Description

This interface gives the API user the possibility to fill the ESA AAL with alarm information during startup of the ESA in order to keep the ESA synchronized with the active alarms in other data sources (subagents) in the system.

Interface: `com.ericsson.esa.fmagent.rmi.ISubagentAALSynchronization`

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`.





7 Administration

7.1 Log Files

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

7.2 Backup and Restore

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





8 Appendix A: Standardized Parameter Values

8.1 Perceived Severity

This chapter contains the part of the MIB “ITU-ALARM-TC” that describes the values of the variable *Perceived Severity*, which is one of the data sent in the SNMP alarm from the ESA.

```
ItuPerceivedSeverity ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "ITU perceived severity values as per M.3100
        and X.733"
    SYNTAX INTEGER {
        cleared (1),
        indeterminate (2),
        critical (3),
        major (4),
        minor (5),
        warning (6)
    }
```

8.2 Event Type

This chapter contains the part of the MIB “IANA-ITU-ALARM-TC-MIB” that describes the values of the variable *Event Type*, which is one of the data sent in the SNMP alarm from the ESA.

```
IANAItuEventType ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "The ITU event Type values.

        The Internet Assigned Number Authority (IANA) is
        responsible for the assignment of the enumerations
        in this TC.

        Request should come in the form of well-formed
        SMI [RFC2578] for enumeration names that are unique
        and sufficiently descriptive.

        See http://www.iana.org "
    REFERENCE
        "ITU Recommendation X.736, 'Information Technology - Open
        Systems Interconnection - System Management: Security
```

```
Alarm Reporting Function', 1992"
SYNTAX INTEGER
{
    other (1),
    communicationsAlarm (2),
    qualityOfServiceAlarm (3),
    processingErrorAlarm (4),
    equipmentAlarm (5),
    environmentalAlarm (6),
    integrityViolation (7),
    operationalViolation (8),
    physicalViolation (9),
    securityServiceOrMechanismViolation (10),
    timeDomainViolation (11)
}
```

8.3 Probable Cause

This chapter contains the part of the MIB “IANA-ITU-ALARM-TC-MIB” that describes the values of the variable *Probable Cause*, which is one of the data sent in the SNMP alarm from the ESA.

```
IANAItuProbableCause ::= TEXTUAL-CONVENTION
    STATUS current
    DESCRIPTION
        "ITU-T probable cause values. Duplicate values defined in
        X.733 are appended with X733 to ensure syntactic uniqueness.
        Probable cause value 0 is reserved for special purposes.

        The Internet Assigned Number Authority (IANA) is responsible
        for the assignment of the enumerations in this TC.
        IANAItuProbableCause value of 0 is reserved for special
        purposes and MUST NOT be assigned.

        Values of IANAItuProbableCause in the range 1 to 1023 are
        reserved for causes that correspond to ITU-T probable cause.

        All other requests for new causes will be handled on a
        first-come, first served basis and will be assigned
        enumeration values starting with 1025.

        Request should come in the form of well-formed
        SMI [RFC2578] for enumeration names that are unique and
        sufficiently descriptive.

        While some effort will be taken to ensure that new probable
        causes do not conceptually duplicate existing probable
        causes it is acknowledged that the existence of conceptual
        duplicates in the starting probable cause list is an known
        industry reality."
```




To aid IANA in the administration of probable cause names and values, the OPS Area Director will appoint one or more experts to help review requests.

See <http://www.iana.org>"

REFERENCE

"ITU Recommendation M.3100, 'Generic Network Information Model', 1995
 ITU Recommendation X.733, 'Information Technology - Open Systems Interconnection - System Management: Alarm Reporting Function', 1992
 ITU Recommendation X.736, 'Information Technology - Open Systems Interconnection - System Management: Security Alarm Reporting Function', 1992"

SYNTAX

INTEGER

```
{
  -- The following probable causes were defined in M.3100
  aIS (1),
  callSetUpFailure (2),
  degradedSignal (3),
  farEndReceiverFailure (4),
  framingError (5),
  lossOfFrame (6),
  lossOfPointer (7),
  lossOfSignal (8),
  payloadTypeMismatch (9),
  transmissionError (10),
  remoteAlarmInterface (11),
  excessiveBER (12),
  pathTraceMismatch (13),
  unavailable (14),
  signalLabelMismatch (15),
  lossOfMultiFrame (16),
  receiveFailure (17),
  transmitFailure (18),
  modulationFailure (19),
  demodulationFailure (20),
  broadcastChannelFailure (21),
  connectionEstablishmentError (22),
  invalidMessageReceived (23),
  localNodeTransmissionError (24),
  remoteNodeTransmissionError (25),
  routingFailure (26),
  --Values 27-50 are reserved for communications alarm related
  --probable causes
  -- The following are used with equipment alarm.
  backplaneFailure (51),
  dataSetProblem (52),
  equipmentIdentifierDuplication (53),
```

```

    externalIFDeviceProblem (54),
    lineCardProblem (55),
    multiplexerProblem (56),
    nEIdentifierDuplication (57),
    powerProblem (58),
    processorProblem (59),
    protectionPathFailure (60),
    receiverFailure (61),
    replaceableUnitMissing (62),
    replaceableUnitTypeMismatch (63),
    synchronizationSourceMismatch (64),
    terminalProblem (65),
    timingProblem (66),
    transmitterFailure (67),
    trunkCardProblem (68),
    replaceableUnitProblem (69),
    realTimeClockFailure (70),
--An equipment alarm to be issued if the system detects that the
--real time clock has failed
    antennaFailure (71),
    batteryChargingFailure (72),
    diskFailure (73),
    frequencyHoppingFailure (74),
    iODeviceError (75),
    lossOfSynchronisation (76),
    lossOfRedundancy (77),
    powerSupplyFailure (78),
    signalQualityEvaluationFailure (79),
    tranceiverFailure (80),
    protectionMechanismFailure (81),
    protectingResourceFailure (82),
-- Values 83-100 are reserved for equipment alarm related probable
-- causes
-- The following are used with environmental alarm.
    airCompressorFailure (101),
    airConditioningFailure (102),
    airDryerFailure (103),
    batteryDischarging (104),
    batteryFailure (105),
    commercialPowerFailure (106),
    coolingFanFailure (107),
    engineFailure (108),
    fireDetectorFailure (109),
    fuseFailure (110),
    generatorFailure (111),
    lowBatteryThreshold (112),
    pumpFailure (113),
    rectifierFailure (114),
    rectifierHighVoltage (115),
    rectifierLowFVVoltage (116),
    ventilationsSystemFailure (117),

```



```

enclosureDoorOpen (118),
explosiveGas (119),
fire (120),
flood (121),
highHumidity (122),
highTemperature (123),
highWind (124),
iceBuildUp (125),
intrusionDetection (126),
lowFuel (127),
lowHumidity (128),
lowCablePressure (129),
lowTemperature (130),
lowWater (131),
smoke (132),
toxicGas (133),
coolingSystemFailure (134),
externalEquipmentFailure (135),
externalPointFailure (136),
-- Values 137-150 are reserved for environmental alarm related
-- probable causes
-- The following are used with Processing error alarm.
    storageCapacityProblem (151),
    memoryMismatch (152),
    corruptData (153),
    outOfCPUCycles (154),
    sfwrEnvironmentProblem (155),
    sfwrDownloadFailure (156),
    lossOfRealTime (157),
--A processing error alarm to be issued after the system has
--reinitialised. This will indicate
--to the management systems that the view they have of the managed
--system may no longer
--be valid. Usage example: The managed
--system issues this alarm after a reinitialization with severity
--warning to inform the
--management system about the event. No clearing notification will
--be sent.
    applicationSubsystemFailure (158),
    configurationOrCustomisationError (159),
    databaseInconsistency (160),
    fileError (161),
    outOfMemory (162),
    softwareError (163),
    timeoutExpired (164),
    underlyingResourceUnavailable (165),
    versionMismatch (166),
--Values 168-200 are reserved for processing error alarm related
--probable causes.
    bandwidthReduced (201),
    congestion (202),

```

```
excessiveErrorRate (203),
excessiveResponseTime (204),
excessiveRetransmissionRate (205),
reducedLoggingCapability (206),
systemResourcesOverload (207 ),
-- The following were defined X.733
adapterError (500),
applicationSubsystemFailure (501),
bandwidthReducedX733 (502),
callEstablishmentError (503),
communicationsProtocolError (504),
communicationsSubsystemFailure (505),
configurationOrCustomizationError (506),
congestionX733 (507),
corruptData (508),
cpuCyclesLimitExceeded (509),
dataSetOrModemError (510),
degradedSignalX733 (511),
dteDceInterfaceError (512),
enclosureDoorOpenX733 (513),
equipmentMalfunction (514),
excessiveVibration (515),
fileErrorX733 (516),
fireDetected (517),
framingErrorX733 (518),
heatingVentCoolingSystemProblem (519),
humidityUnacceptable (520),
inputOutputDeviceError (521),
inputDeviceError (522),
lanError (523),
leakDetected (524),
localNodeTransmissionErrorX733 (525),
lossOfFrameX733 (526),
lossOfSignalX733 (527),
materialSupplyExhausted (528),
multiplexerProblemX733 (529),
outOfMemoryX733 (530),
ouputDeviceError (531),
performanceDegraded (532),
powerProblems (533),
pressureUnacceptable (534),
processorProblems (535),
pumpFailureX733 (536),
queueSizeExceeded (537),
receiveFailureX733 (538),
receiverFailureX733 (539),
remoteNodeTransmissionErrorX733 (540),
resourceAtOrNearingCapacity (541),
responseTimeExcessive (542),
retransmissionRateExcessive (543),
softwareErrorX733 (544),
```



```
softwareProgramAbnormallyTerminated (545),
softwareProgramError (546),
storageCapacityProblemX733 (547),
temperatureUnacceptable (548),
thresholdCrossed (549),
timingProblemX733 (550),
toxicLeakDetected (551),
transmitFailureX733 (552),
transmitterFailure (553),
underlyingResourceUnavailable (554),
versionMismatchX733 (555),
-- The following are defined in X.736
authenticationFailure (600),
breachOfConfidentiality (601),
cableTamper (602),
delayedInformation (603),
denialOfService (604),
duplicateInformation (605),
informationMissing (606),
informationModificationDetected (607),
informationOutOfSequence (608),
keyExpired (609),
nonRepudiationFailure (610),
outOfHoursActivity (611),
outOfService (612),
proceduralError (613),
unauthorizedAccessAttempt (614),
unexpectedInformation (615),

other (1024)
}
```





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 SNMP Interface for Fault Management*
INTERWORK DESCRIPTION, 4/155 19-CSH 109 532