SKA LOGGING GUIDELINES

Document number	XXX
Revision	A
Authors	S. Riggi M. Di Carlo E. Giani L. van den Heever
Date	2016-06-30
Status	Draft

Document History

Revision	Date Of Issue	Description
Draft	2016-06-30	First draft prepared by the Logging task team for LMC Harmonisation meeting in Edinburgh (4-6 Jul 2016)
A	XXX	TBD Document passed revision of Tango experts
B	XXX	TBD Document passed internal SKAO and Ant team revision
C	XXX	TBD Document passed final revision of LMC teams

Table of Contents

Referenced Documents

[RD1] M. Di Carlo, SKA1 TM LMC LOGGING SERVICE NOTE
[RD2] The TANGO Team, *The TANGO Control System Manual* - Version 9.2
[RD3] Rsyslog: rocket-fast system for log processing, http://www.rsyslog.com
[RD4] BSD Syslog protocol RFC 3164, https://tools.ietf.org/html/rfc3164
[RD5] Syslog protocol RFC5424, https://tools.ietf.org/html/rfc5424
[RD6] Apache log4cxx, https://logging.apache.org/log4cxx/
[RD7] Log4cpp, http://log4cpp.sourceforge.net/
[RD8] Log4cplus, https://github.com/log4cplus/log4cplus
[RD9] Boost.Log, http://www.boost.org/
[RD10] Google gLog, https://github.com/google/glog
[RD11] http://www.pantheios.org/performance.html
[RD12] Elasticsearch, https://www.elastic.co/products/elasticsearch
[RD13] Logstash, https://www.elastic.co/products/logstash
[RD14] Kibana, https://www.elastic.co/products/kibana
[RD15] Tango Device Servers Implementation Guidelines - Design & Implementation Guidelines, Rev 6, ftp://ftp.esrf.eu/pub/cs/tango/tangodesignguidelines-revision-6.pdf
[RD16] LMC Ant team and SKAO, *SKA Control System Guidelines*
[RD17] LSR: SKA-TEL-TM-0000030-TM-TELMGT-GDL-Rev01_LMC_Scope_and_Responsibilities
[RD18] LIG: SKA-TEL-TM-0000031-TM-TELMGT-GDL-Rev 01_LMC_Interface_Guidelines
[RD19] https://docs.mongodb.com/ecosystem/use-cases/storing-log-data

Glossary

Term	Description
LogConsumer	A special Tango device server implementing a so-called LogConsumer interface as described in [RD2], Section A9, and reviewed in Section 3.1.1 of this document.
ElementLogger	A given Element could deploy a hierarchy of LogConsumer Tango devices in its architecture. The term *ElementLogger* is used to refer to the top-level LogConsumer within the Element.
CentralLogger	The term *CentralLogger* is used to refer to the central Tango LogConsumer at TM that collates logs across all Tango facilities in the telescope.

1. Introduction

Discuss level of standardization to be analyzed.

The design of the SKA Control system recently started a harmonization phase including:

- Refinement of key concepts (e.g. drill-down, rolled-up reporting, TM-LMC control and monitoring interactions, high-level control system organization, etc) initially defined in the LSR [RD17] and LIG [RD18] documents after the choice of the Tango Framework [Note: these documents will be superseded by the SKA Control System Guidelines [RD16] and the relevant contents adopted in the harmonisation work.]
- Standardization of major control system components across all LMCs according to a given priority level agreed by all LMCs. The identified areas of standardization and prioritized items are specified in [RD16].

The standardization of the logging system was given a high-priority at the 2nd Harmonization meeting (Madrid, Apr 2016) and a dedicated task force has been selected to tackle this issue.

The present document aims at exploring and defining the standards and patterns for the Tango logging system to be adopted by all SKA LMCs and addressing all aspects related to logging in the LMC-TM interface. In Table 1.1. we report a list with aspects/issues/patterns to be defined.

Table 1.1: List of logging issues to be discussed/addressed/finalized

Issue ID	***Description***
1	**Logging Architecture** Define a standardized LMC logging architecture, including: ● Organization of logging Tango devices within the LMC Control systems ● Strategy for log reporting to TM ● Strategy for local log archiving ● Strategy for retrieval/backup/sync of log data for archiving purposes centrally
2	**Log format** Which format to be adopted for the log messages generated by Tango devices compliant with the information requested by TM?
3	**Logging Configuration** Define the logging targets and log level threshold to be used under normal and “drill-down/inspect” operations. Address the following issues: ● How to configure and set different logging targets with different levels in Tango devices, e.g. streaming and archiving at different levels ● Do we allow log streaming from LMC Local Logger to the TM Central Logger and at what level? And how to activate/deactivate this? ● Configuration file (i.e. log4j) and log forwarding (i.e. rsyslog/logstash): Do we require standardized template files with mandatory information (e.g. endpoint configuration, custom filters, etc)?
4	**Log Archiving** If archiving to file is selected, define: ● *Policy of log rotation and persistence:* Which policy to be adopted for log file rotation and persistence of archived logs inside LMCs? ● *Multiple vs Single File***:** Tango devices can log to files (either using Tango native schema or

	syslog). Do we require archiving to a single file or to multiple files (e.g. a file per each Tango device) If another log archive is selected (like a database) then define how long logs should be kept and at what time it can be deleted.
5	**Time synchronization** It is very important for all systems reporting logs to be using the same time server (e.g. NTP) so that logs are all synchronized with a good accuracy. Logs (but also alarms/events/monitoring data) shall be also timestamped in a standardized format in SKA. UTC is the proposed format. On the basis of precursors' experience it is suggested to not directly set machine time in UTC, but rather leave the local timezone and set/convert timestamp in UTC in the application generating the log.
6	**Log Visualization/Inspection** Define strategy for visualizing/browsing and inspecting logs. Which architecture and technologies to be supported by LMCs to support central log visualization and inspection in TM? For example, each Element can provide a local configurable LogConsumer collecting all logs from the system to support a LogViewer instance to be run locally or remotely in TM. Define how this device is known to TM (e.g. through naming convention, ...) and how viewers should be configured to allow live log viewing from the Element startup phase. Define features to be provided by LMCs to support viewing/inspection operations in TM beyond the Tango LogViewer, e.g. searching, filtering, log plots, etc.

The document is organized as follows. Section 2 is dedicated to a short review of the logging systems adopted in existing radio observatories or SKA precursors. Section 3 presents an overview of suitable common logging technologies to be taken under consideration by LMCs for the development of their control system prototype. Section 4 discusses possible logging architecture models to be considered at LMC and TM level on the basis of current logging requirements (reported in Section 4.1). Section 5 is devoted to final guidelines and prescriptions delivered to SKA. Section 6 presents the prototyping activities performed to define the guidelines.

2. Review of telescope logging systems

Report here a short review on logging systems adopted in existing radio observatories

The following sections contain a short review of the logging systems in use in existing radio telescope facilities.

2.1 ASKAP

The Australian SKA Pathfinder uses a relational database (table) to log data from all over the system with the help of the Apache Logging library (log4php, log4j, ...) as software library. The log informations are kept for a maximum of a week in the database before being discarded; the reason why has to be found in the architectural design of the control system: the project works with several monitoring points (around 800k) with some machine learning algorithm to analyse them.
However real time logging is available if needed but there is no way to gather data like statistics from the logging service.

2.2 Meerkat

The South African 7-dish Karoo Array Telescope (KAT-7) and the forthcoming Square Kilometre Array (SKA) precursor, the MeerKAT Telescope, is composed by different processes which produces log informations and send them via a central logger.
Standard python logging configuration and the python log handler are used to send each log via TCP to the central logger which gathers and flush out all the logs to files (every file identifies a logger which the processes setup and can share).
Typically each process logs to its own file and it is possible to have specific shared log files for specific interactions like for instance user interactions, alarms, system activity and so on and it is also possible to log the low level protocol messages when required.
The operators and engineers hardly look into logs older than a week but the files are kept for a year.
There is no optimization for any type of search but the central logger gives a view of the current logging that can be filtered by log file, and log levels. Searching is typically performed with OS level text file search tools like grep and this approach has been found to be sufficient thus far.
An example of log format is shown in the following table:

Date Time	Sender name	LEVEL	Message
2015-10-25 23:40:31.123Z	kat.m063	INFO	set_mode(mkat_receptor_proxy.py:977) Setting mode to 'STOP'.

2.3 ALMA

The Atacama Large Millimeter/sub-millimeter Array (ALMA) write log informations in Xml format with message written in natural language. The use of logs is extensive in order to gather informations from the system and having a high level knowledge on it performing data mining activities.
The ALMA architecture is based on ElasticSearch [RD12]/LogStash [RD13]/Kibana [RD14] platform which allow both data mining activities and simple queries (web based). Data mining is used to provide statistics and to detect well known failures together with the ability to provide statistics to give some feelings about how was the observation of the previous night.
Usually data are kept for 24 months with the help of apache Lucene for the partitioning the data.

3. Suitable Logging Technologies for SKA

Describe here the technologies to be employed inside LMCs (Log4J libraries, etc...) and in TM (e.g. ElasticSearch, ...).

Since there are several valid possibilities on the market, there is no need for rebuild a logging system which is already available. In this section we therefore report a short review of present logging technologies that can be employed by LMCs for their prototype development to manage logs from Tango devices and system services.

- *Tango Logging Service (TLS)*
- *Rsyslog*
- *Syslog-ng*
- *Elasticsearch/Logstash/Kibana stack*
- *MongoDB*

Note that some of these could also be employed for central logging by TM.

3.1 The Tango Logging Service (TLS)

Tango incorporates a Logging Service, called the Tango Logging Service (TLS) or log4tango, which is based on the

logback library (a branch of Log4j) for Java devices and on a custom Log4j-like implementation for C++ and python devices.
TLS allows device messages to be given the following ordered log levels (semantic is just given to be indicative of what could be logged at each level):

DEBUG < INFO < WARN < ERROR < FATAL < OFF

- OFF: Nothing is logged
- FATAL: A fatal error occurred. The process is about to abort
- ERROR: An (unrecoverable) error occurred but the process is still alive
- WARN: An error occurred but could be recovered locally or is not critical impacting functionality
- INFO: Provides information on important actions performed
- DEBUG: Generates detailed information describing the internal behaviour of a device

The level acts as a filter to control the information sent to the targets. For a given device, a level is said to be enabled if it is greater or equal to the logging level assigned to this device. In other words, any logging request whose level is lower than the device's logging level is ignored and not sent to the targets.
Logs can be sent simultaneously to three multiple logging targets:

1. *CONSOLE*: log displayed on a console (the classical way)
2. *FILE*: logs are sent and stored in a XML file
3. *DEVICE*: logs sent to a Tango device called LogConsumer, implementing the Tango interface described in ref. R2 (Appendix 9)

Multiple entries can be specified for each target category, e.g. it is possible to have two or more Tango devices as logging targets.
Additionally, a graphical application (a generic implementation of the log consumer device), named ***LogViewer*** (see section 3.1.1 for details and Fig. 3.1.1 for an application screenshot), allows to register (set a logging level and a logging target) a device for logging purpose (automatically from the database testing tool – Jive – or directly from the software), view and filter the log messages generated by the attached devices on the basis of a string filter (for date, for the type of class of the device, etc.). *This means that the framework provides, for each device server, a set of api that add a generic listener of the log messages.*

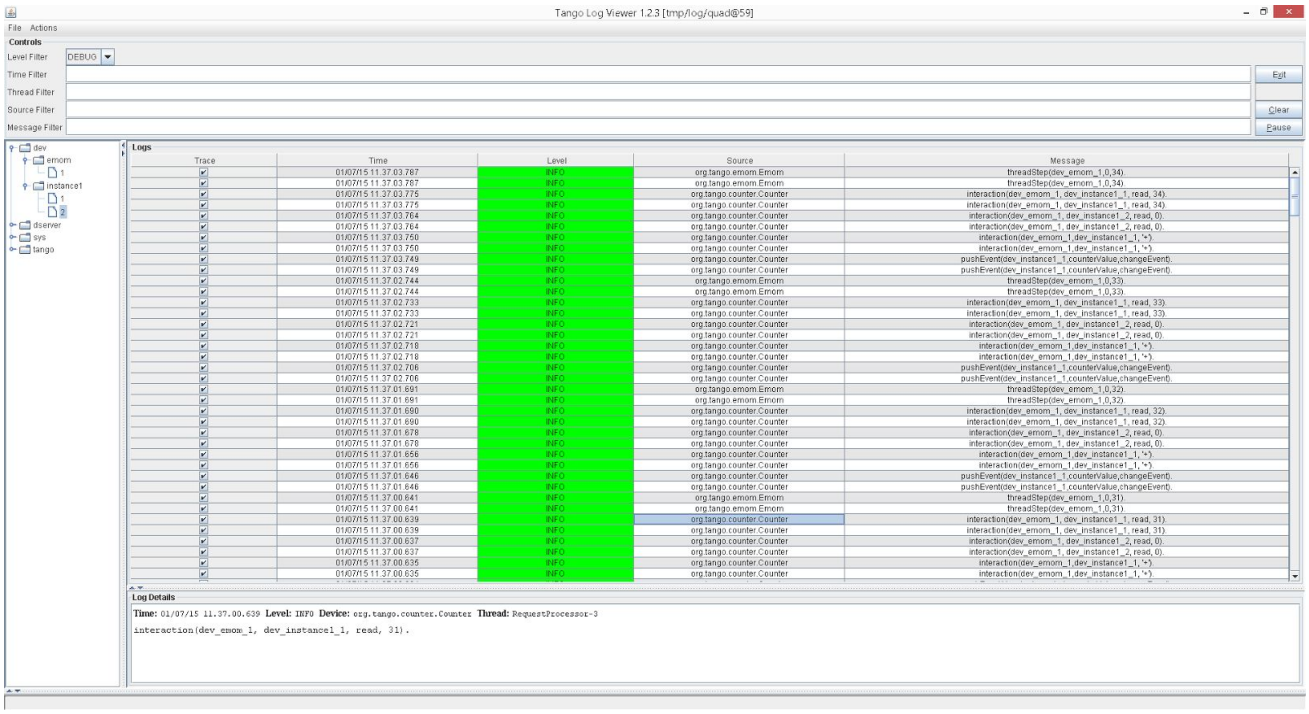

Fig. 3.1.1: Standard Tango Log Viewer

The device's logging behaviour can be controlled by adding and/or removing targets and level using the builtin *dserver* admin commands. If multiple devices are registered within the same server, it is possible to specify different log targets/levels for each one of them.
By default the assigned log level is equal for all log appenders specified (e.g. FILE, CONSOLE, DEVICE targets) as explicitly quoted in the Tango Manual:

Note: The logging level can't be controlled at target level. The device's targets shared the same device logging level.

However, if desired, different logging levels for appenders can be selected by enabling the APPENDERS_HAVE_LEVEL_THRESHOLD flag when building the Tango Control source code (see the prototype section for an example) .

3.1.1 The Log Consumer device

The log consumer device can be described as a Tango device corresponding to the UML model shown in Figure 3.1.2.

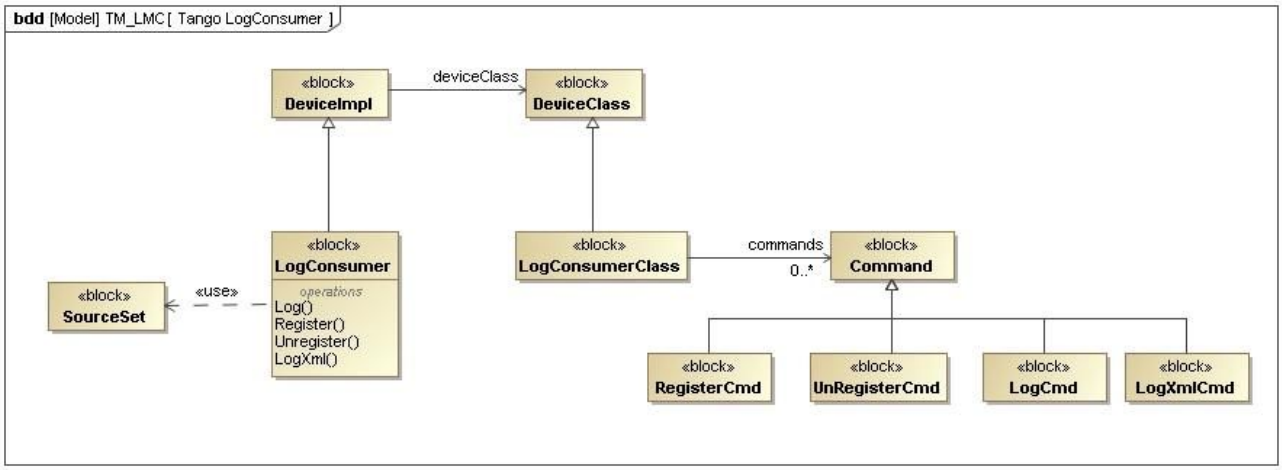

Fig. 3.1.2: The Log consumer device UML model

The device has four commands:

- Register: used to register a target (see 2) to the log consumer for logging purpose;
- Unregister: used to unregister a target (see 2);
- Log: log a message from a target (the message is an array of strings);
- LogXml: log a message from a target (the message is an xml string).

The LogViewer application instantiates a LogConsumer which is configured as a logging target for the devices the LogViewer registers for logging.

Figure 3.1.3 shows the device from the Tango test application available from The Tango Jive application.

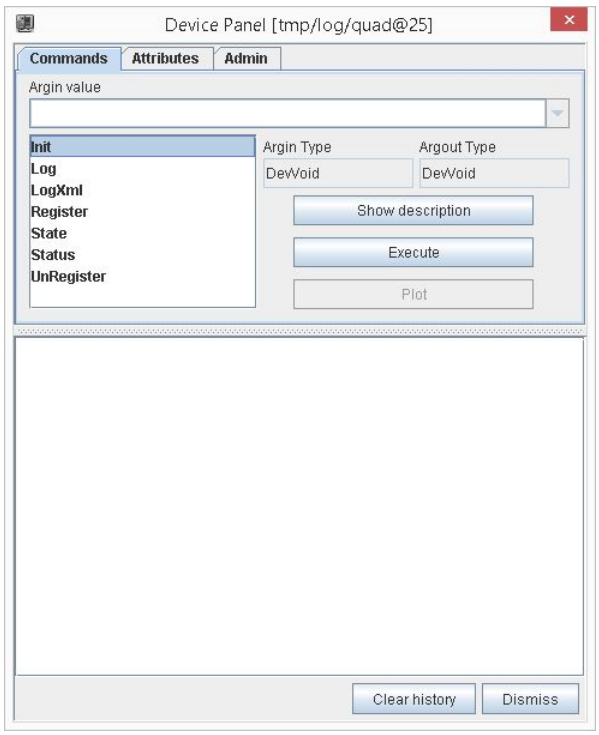

Fig. 3.1.3: A log consumer device

3.2 Rsyslog

Rsyslog (**r**ocket-fast **sys**tem for **log** processing, see [RD3]) is a Log Manager for Unix systems. It is very scalable, widely available in many Linux distributions and the default logging system in some of them. It allows to:

- be started as a regular syslogd daemon, being an advanced version of the standard sysklogd
- accept log inputs from a wide variety of sources (e.g. typically TCP/UDP connections are used by Log4j-like libraries) with restrictions imposed (e.g. on port/ip). See Fig. 3.2.1 and rsyslog manual for an updated list of supported input modules.
- Support different protocols beyond the BSD standard syslog protocol RFC 3164 [RD4], e.g. RFC 5424 [RD5], RFC 5425, RFC 5426, but also ISO 8601 timestamp with millisecond granularity and timezone information, GSS-API and TLS, etc.
- transform/filtering the received logs, e.g. discriminate the logs by program, source, message, pid
- locally buffer the input logs in case the log destination is not ready
- output the log transformed results to different destinations, for example a remote rsyslog server, search engines (e.g. Elasticsearch), or data store (e.g. mongodb). See Fig. 3.2.1 and rsyslog manual for an updated list of supported output modules.

See the reference manual for a complete list of all supported features.

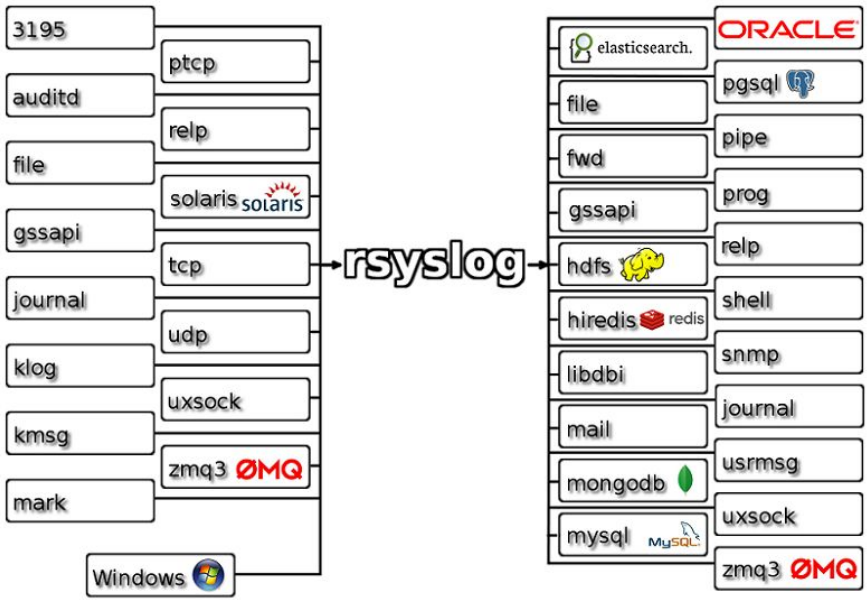

Fig. 3.2.1: *Overview of input and output source supported by rsyslog (taken from [RD3])*

According to the default BSD standard syslog protocol RFC 3164 [RD4] the full format of a syslog message sent on the wire has three discernable parts with total length 1024 bytes or less (see Fig. 3.2.2):

1. **PRI**: The *Priority* part is a 8-bit number enclosed in angle brackets <> and representing both the message *Severity* (i.e the log level with the first 3 least significant bits, thus up to 8 different severities) and the *Facility* (i.e. the source application generating the log (the remaining 5 bits). Predefined codes for facility and severity levels are reported in Table 5.1.3.1 and 5.1.3.2. The log priority level can then be computed from severity and facility in a straightforward way.

2. **HEADER**: The HEADER part contains two fields:
    a. *TIMESTAMP* (date & time) at which the message was generated, generally taken from the system time
    b. *HOSTNAME* (or ip address) of the server generating the message
3. **MSG**: The MSG part has two fields:
    a. *TAG*, i.e. the name of the application/process that generated the message with a length not exceeding 32 characters
    b. *CONTENT* field, i.e. the detail of the message

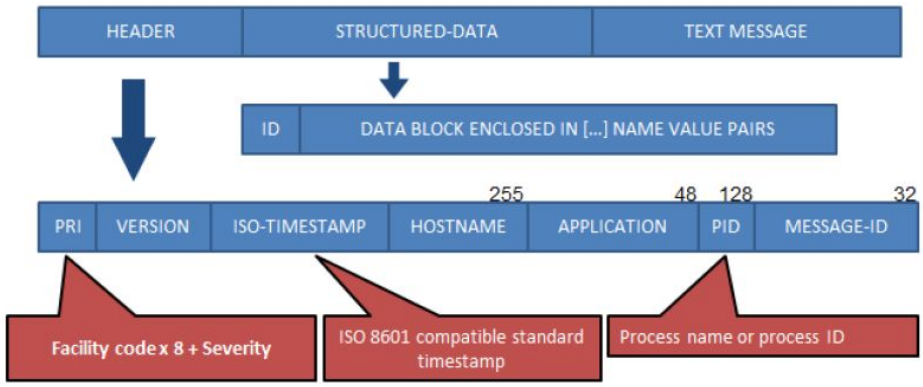

Fig. 3.2.2: The syslog message format

Table 3.2.1: *List of syslog facility codes as described in protocol RFC3164*

Facility Number	Keyword	Facility Description
0	kern	kernel messages
1	user	user-level messages
2	mail	mail system
3	daemon	system daemons
4	auth	security/authorization messages
5	syslog	messages generated internally by syslogd
6	lpr	line printer subsystem
7	news	network news subsystem
8	uucp	UUCP subsystem
9		clock daemon
10	authpriv	security/authorization messages
11	ftp	FTP daemon
12	-	NTP subsystem
13	-	log audit
14	-	log alert
15	cron	clock daemon
16	local0	local use 0 (local0)
17	local1	local use 1 (local1)

18	local2	local use 2 (local2)
19	local3	local use 3 (local3)
20	local4	local use 4 (local4)
21	local5	local use 5 (local5)
22	local6	local use 6 (local6)
23	local7	local use 7 (local7)

Table 3.2.2: *List of syslog severity codes as described in protocol RFC3164 and a possible mapping to Tango log levels*

Syslog Code	Syslog Severity	Description	Tango Code	Tango Level
-	-	-	0	OFF
0	Emergency	System is unusable	1	FATAL
1	Alert	Action must be taken immediately	1	FATAL
2	Critical	Critical conditions	1	FATAL
3	Error	Error conditions	2	ERROR
4	Warning	Warning conditions	3	WARNING
5	Notice	Normal but significant condition	3	INFO
6	Informational	Informational messages	4	INFO
7	Debug	Debug-level messages	5	DEBUG

The new syslog protocol RFC5424 [RD5] is backward compatible with RFC3164[1] and defines additional message fields in the header:

- *VERSION*: The syslog protocol version, i.e. 1 for RFC5424
- *APP-NAME*: A string marking the device or application that originated the message, mainly intended for filtering messages
- *PROCID:* process name or process ID associated with a syslog system
- *MSGID:* type of message
- *STRUCTURED-DATA:* A list of SD-ELEMENT field consisting in name (SD-ID) and parameter name-value pairs (SD-PARAM)

3.3 Syslog Logging Libraries

For SKA needs it is preferable to employ existing and tested logging libraries supporting syslog rather than developing new software to be maintained. In this section we therefore report on the best solutions we have found for different languages.

[1] Note that the timestamp field has some restrictions with respect to the previous format (see RFC5424 specs for details)

3.3.1 C++ Loggers

Ex: Log4Cxx, Log4Cplus, Log4Cpp, Boost.log, gLog, ...

There are several logging libraries in C++ providing support for syslog. Among them, the most used are: Log4Cxx [RD6], Log4Cplus [RD7], Log4Cpp [RD8], Boost.log [RD9], gLog [RD10]. All libraries are fully thread-safe. The first three are derived from log4j and thus have an API similar to the C++ Tango logging component.
All of them support the syslog protocol version RFC 3164 [RD3]. Plans to support the new syslog protocol RFC 5424 [RD4] was announced by Boost.log back in release 1.54 and also in current release 1.61. We therefore do not expect to have support for it for the development of the SKA LMC prototypes.
A comparison of the relative performances for Log4Cxx, Log4Cplus, Log4Cpp, Boost.log is presented in [RD11] in the context of the Pantheios logging library (which was claimed the fastest one). Boost.log performs slightly better compared to log4-like libraries in some of the scenarios tested but overall their performances are comparable.
From the release date and frequency of activity/commits in the repository Boost.log, Log4Cplus and gLog seem the most actively developed at present. Latest Log4Cxx release (0.10.0) for example is dated 2014-02-22 and seem unmaintained.
The available documentation is poor for the all libraries explored. It is thus instructive to report how to configure and send logs to syslog for some of them: Log4cxx and Boost.

Log4cxx

```
// Define static logger variable
log4cxx::LoggerPtr  logger(log4cxx::Logger::getLogger(device_name.c_str()));

//Define the layout used to format the log message
// %c: logger name
// %d: date format (ISO8601 if not given)
// %F: log source file name
// %l: caller location information
// %L: log line number
// %m: application name
// %n: line separator char
// %p: log level
// %r: milliseconds from app start to this log
// %x: NDC (nested diagnostic context)
// %X: MDC (mapped diagnostic context), example %X{myproperty}
log4cxx::LayoutPtr layout( new log4cxx::PatternLayout("%m%n") );

//Create the syslog appender
log4cxx::AppenderPtr appender( new log4cxx::net::SyslogAppender(layout, host, facility) );

//Add syslog appender to logger
logger->addAppender(appender);

//Set the desired logging level
std::string level= “INFO”;
logger->setLevel( log4cxx::Level::toLevel(level) );

//Send an example info log
LOG4CXX_INFO (logger, "An info message”);

//Alternatively use log methods for setting log location fields or defining user macros
//logger->log(LevelPtr level, const std::string &message, const log4cxx::spi::LocationInfo &location)
```

Boost.log

```
namespace logging = boost::log;
```

```
namespace src = boost::log::sources;
namespace sinks = boost::log::sinks;
namespace keywords = boost::log::keywords;

//Init
boost::shared_ptr< logging::core > core = logging::core::get();

//Create a syslog backend
boost::shared_ptr<sink_t> sink(
  new sink_t(
     keywords::use_impl = sinks::syslog::native,  //native or udp_socket_based
     keywords::facility =  facility_code,         //e.g. sinks::syslog::local6
     keywords::ident = device_name                //the device name
   )
);

//Set log format
sink->set_formatter(
  boost::log::expressions::format("%1%")
    % boost::log::expressions::smessage
);

//Set log level mapping from boost enum to syslog
enum boost_severity_level{
  boost_off= 0,
  boost_fatal= 1,
  boost_error= 2,
  boost_warn= 3,
  boost_info= 4,
  boost_debug= 5
};
BOOST_LOG_ATTRIBUTE_KEYWORD(severity,"Severity",boost_severity_level)

sinks::syslog::custom_severity_mapping<boost_severity_level> mapping("Severity");
mapping[boost_info]= sinks::syslog::info;
mapping[boost_warn]= sinks::syslog::warning;
mapping[boost_debug]= sinks::syslog::debug;
mapping[boost_error]= sinks::syslog::error;
mapping[boost_fatal]= sinks::syslog::critical;
sink->locked_backend()->set_severity_mapper(mapping);

//Set local & target address
sink->locked_backend()->set_local_address(syslog_host);
sink->locked_backend()->set_target_address(syslog_host);

//Set log level threshold
boost_severity_level filter_level= boost_info;
sink->set_filter(severity <= filter_level);

// Add the sink to the core
logging::core::get()->add_sink(sink);

//Send an example info log
src::severity_logger_mt<boost_severity_level> lg(
  keywords::severity = boost_info
);
BOOST_LOG_SEV(lg, boost_info) << "An info message";
```

3.3.2 Java Loggers

Discuss java logger libraries for syslog

The TANGO framework is based on a library called *logback* (a branch of the log4j library). Every Device Server in the Tango System, has two public methods: the "AddLoggingTarget" and "setLoggingLevel".

Depending on the message sent, the "AddLoggingTarget" method add, to the logback configuration, a new appender that can be a DeviceAppender or a FileAppender. While the FileAppender is a standard feature of logback, the DeviceAppender is a specific extension to the library to support sending log informations over a device server called LogConsumer (usually related to the usage of the tango tool LogViewer).
The "setLoggingLevel" method set the logging level of the appender (only the level of the appender with the same name of the device).
The logging target are appender and the logging level is set at appender level. This means that other appenders, with different names, are not touched by the tango mechanism.
The Syslog is a standard feature for logback and to configure it, it is necessary to add an xml configuration file (named logback.xml) with the correct configuration, like:

```
<appender name="SYSLOG" class="ch.qos.logback.classic.net.SyslogAppender">
    <syslogHost>remote_home</syslogHost>
    <facility>AUTH</facility>
    <suffixPattern>[%thread] %logger %msg</suffixPattern>
</appender>
```

In order to have syslog into file, it is necessary to extends the logback library with a SysLogFileAppender[2] that does not send log messages over the network but write it into a file.

3.3.3 Python Loggers

Discuss python logger libraries for syslog

Logging to syslog in python Tango devices could be performed by using the SysLogHandler class provided by the python logging library:

```
import logging
import logging.handlers

logger = logging.getLogger('myLogger')
logger.setLevel(logging.INFO)

#add handler to the logger
handler = logging.handlers.SysLogHandler('/dev/log')

#add formatter to the handler
formatter = logging.Formatter('Python: { "loggerName":"%(name)s", "asciTime":"%(asctime)s",
             "pathName":"%(pathname)s", "logRecordCreationTime":"%(created)f",
             "functionName":"%(funcName)s", "levelNo":"%(levelno)s", "lineNo":"%(lineno)d",
             "time":"%(msecs)d", "levelName":"%(levelname)s", "message":"%(message)s"}')

handler.formatter = formatter
logger.addHandler(handler)

logger.info("An info message")
```

[2] The TM.LMC team has already developed it. Refer to https://skatelmgr.atlassian.net/browse/MON-1733

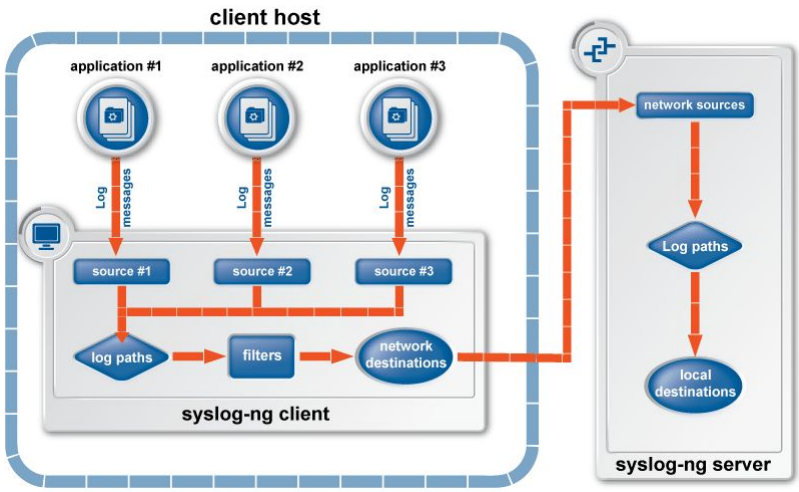

Fig. 3.4.1: syslog-ng client and server

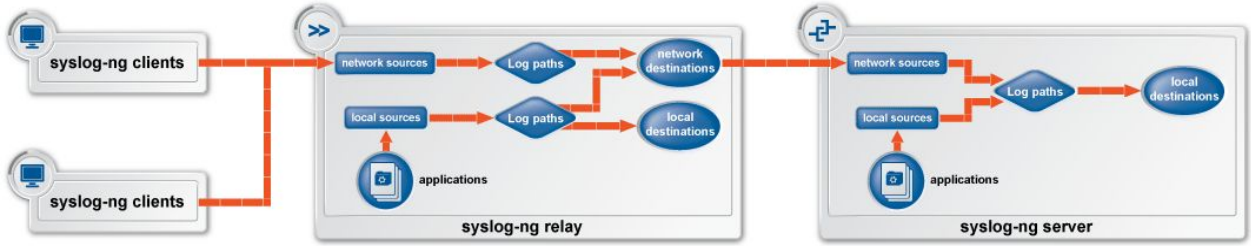

Fig. 3.4.2: syslog-ng client, relay and server

3.4 Syslog-ng

Syslog-ng is a scalable system logging for a centralized logging solution based on three types of entities: the server, the client and the relay.
Within a client host there is an agent which collect every log messages from various applications and/or devices. In the client configuration there can be some global object like log paths, filters, network destinations and so on. The log path is used to connect source to destination and it can include one or more filter (usually regular expression used to select messages), a parser (for instance a json parser to store the log informations in a Mongo DB server) and rewriting rules (to completely change a message into another format). There are many possible destinations within this product like relational database, non-relational DB, ElasticSearch server, plain files and many others.
The syslog-ng architecture is summarized in Figure 3.4.1 and 3.4.2 in a logical schema.
There are different best practises associated to the syslog-ng application. One of the most important is the NTP synchronization on all clients and on the syslog server. It is very important for all systems reporting logs to be using the same time server so that logs are all synchronized with a good accuracy.
There are also other best practise and possibilities including:

- the possibility of using a secure socket layer;
- group “like sources” into the same log file;
- include logs and log archives in a standard backup process for disaster recovery;
- Change read/write permission on log files so that they are not accessible to unprivileged user accounts.

3.5 Elasticsearch/Logstash/Kibana stack

One of the requirements for a good logging service is to have a fast search in order to analyse big volumes of data as quickly as possible (or better in near real time). *Elasticsearch* solves the problem with a full-text search and an analytics engine.
At the core of Elasticsearch there is *Apache Lucene* which is a free and open-source information retrieval software library. While suitable for any application that requires full text indexing and searching capability, Lucene has been widely recognized for its utility in the implementation of Internet search engines and local, single-site searching.
At the core of Lucene's logical architecture is the idea of a document containing fields of text. This flexibility allows Lucene's API to be independent of the file format. Text from PDFs, HTML, Microsoft Word, Mind Maps, and OpenDocument documents, as well as many others (except images), can all be indexed as long as their textual information can be extracted.
Logstash is designed for collecting, aggregation, parsing data and putting them into Elasticsearch in order to run searches and aggregations to mine any information of interest.
Kibana is designed for analytics/business-intelligence needs, to quickly investigate, analyse, visualize, and ask ad-hoc questions on a lot of data (millions or billions of records). In fact it is possible to build custom dashboards that can visualize aspects of the data that are important.
According to the logging architecture described in Section 4.3, Elasticsearch is the data centre, Logstash is the forwarder and Kibana is a visualizer engine (the globe in Figure 4.3.1).
According to Elasticsearch documentation[3], there are different concepts that need to be understood to realize the product.
The basic unit of information which can be indexed is a document that is expressed in JSON[4]. An index is a collection of documents that have similar characteristics; within an index, it is possible to define one or more types which are a logical category (or partition) of your index (the semantic is up to the designer). Since log data can be considered as big data, an index can store a large amount of them and exceed the hardware limits of a single node[5]. It is possible to subdivide the index into multiple pieces called shards. When defining an index it is important to define the number of shards as well for two main reasons: to scale the content volume horizontally and to distribute and parallelize operations across shards which will result in increasing performance/throughput. It is also possible to have replica shards (a copy of a shard) to provide high availability in case a node/shard fails and to scale out the search volume/throughput since they can be executed on all replicas in parallel.
Elasticsearch provides a very comprehensive and powerful REST API to interact with the cluster for any type of operations.

3.5.1 Kibana

Kibana is an open source analytics and visualization platform designed to work with Elasticsearch. It provides visualization capabilities on top of the content indexed on an Elasticsearch cluster.
It is composed of three sections (Discover, Visualize, Dashboard) and defines each set of data loaded in Elasticsearch based on an Index pattern. The Discover view presents all the data in an index pattern as a table of documents. It is also possible to execute query, filter, and so on. Visualization is the heart of Kibana. In this section it is possible to define aggregation and visualization of data obtained using queries defined in the Discovery section in different ways, for instance line chart, bar chart, pie chart and so on. A Dashboard view is a place where to aggregate several graph defined in Visualization view.

[3] https://www.elastic.co/guide/en/elasticsearch/guide/current/index.html
[4] Javascript Object notation: http://www.json.org/
[5] A node is a single server that is part of a cluster; the latter is a collection of one or more servers that together form your data centre.

Fig. 3.5.1: An screenshot of Kibana Dashboard

3.6 MongoDB

The basic element is the Document that is a record in a MongoDB *collection* and the basic unit of data in MongoDB. Documents are analogous to *JSON* objects but exist in the database in a more type-rich format known as *BSON*.

When using this technology it is important to extract log data into individual fields in a document because only in this way log data can be useful for the developer or maintainer.

In particular, what is important to reach is an high throughput (write concern) and a good management of the data growth.

With MongoDB, the write concern are treated in four possible ways:

1. No acknowledgement: the fastest option, but also the unsafe one;
2. Simple acknowledgement: ensure that MongoDB acknowledges inserts;
3. Acknowledgement to Disk: MongoDB not only *acknowledge* receipt of the message but also commit the write operation to the on-disk journal
4. Acknowledgement to Replica: MongoDB replicate the data to multiple *secondary* *replica* *set* members before returning.

MongoDB introduces also the concept of sharding that is the ability to distribute data among two or more DB instances (horizontal partition). The sharding is based on a shard key and choosing correctly the key is very important because it will affect the writing speed: using, for instance, a time based key will slow down the insert as choosing a random key.

It is also very important to choose a strategy for the data growth:

- Capped collection: has a fixed size, and drop old data when inserting new data after reaching cap (no shard possible);
- Multiple Collections, Single Database: Periodically rename your event collection so that your data collection rotates in much the same way that you might rotate log files. When needed, you can drop the oldest collection from the database;
- Multiple Databases: Rotate databases rather than collections.

More information on MongoDB logging system are available in [RD19].

4. SKA Logging Guidelines

Describe here the SKA logging guidelines defined for SKA.

4.1 LMC Logging Requirements

Collect and report here all the logging requirements currently given to LMCs by SE teams and requirements given by TM. Report here a list of issues to be investigated.

We collected in Table 4.1.1 all requirements related to logging given to SKA Elements by the Consortia SE teams.

Table 4.1.1: List of general logging requirements given to SKA Elements

Element	*Req. ID*	*Description*
TM.LMC	TM.LMC_REQ_083	**Logs Database** LMC shall maintain TM logging data in a central repository
	TM.LMC_REQ_085	**Archive log messages** LMC shall time stamp and store all log messages in the archive with TBD precision
	TM.LMC_REQ_086	**Archive alarms, warning and errors** LMC shall log and archive all reported alarms/warnings/errors for at least TBD months
	TM.LMC_REQ_140	**Logging access** LMC shall provide: 1. functionalities to manage logging from remote sites 2. functionalities to manage logging from local sites
	TM.LMC_REQ_141	**Logging content** LMC shall log: 1. each event (including alarms) that has been triggered 2. each high level command that has been received 3. each low level command that has been sent 4. each attempt of metering configuration 5. each attempt of monitoring configuration 6. each response that has been sent or received 7. each false positive fault that has occurred 8. each false positive failure that has occurred 9. each result of a failure prediction that has occurred 10. the health of system components 11. notes on troubleshooting and maintenance activities 12. all configuration changes 13. TBD
	TM.LMC_REQ_142	**Generate log report** LMC shall generate a log report for a specific period of time consisting of all relevant system logs and user logs on request
	TM.LMC_REQ_143	**Generate management report** LMC shall generate management report including at least: 1. Components functionality efficiency 2. Number of faults 3. Time loss logs
	TM.LMC_REQ_301	**Search logging files** The TM shall allow users and operator with authorization to access and search for log files through words, datetime and any other elements the operators request
	TM.LMC_REQ_302	**Performance in logging** Log files shall not influence the main communication channel of the network
	TM.LMC_REQ_303	**Active logging** The LMC shall allow to view and interrogate log informations while logging is active

	TM.LMC_REQ_304	**Log information** The log shall unambiguously show the source of each message like thread, process, module and so on.
DSH.LMC	R.LMC.FMON.12	**LMC Report Logs** The LMC shall report all log messages for DSH to TM, and shall allow TM to control logging reporting, including: a. The destination for logging messages b. The logging level.
	R.LMC.FMON.19	**LMC Reporting on missing components** LMC shall not report alarms, events, logs, or faults on missing DSH components
	R.LMC.FMON.25	**LMC Remote access of logging files** LMC shall allow TM to access and copy local (to the DSH) logging files (where applicable) TBC.
CSP.LMC	SKA1-CSP-LMC-REQ-2279-01-00	**CSP Element Log File** CSP_LOW.LMC and CSP_Mid.LMC shall maintain CSP Log Files, in plain text format, that record internal warnings na errors that occur down to the LRU levels.
	SKA1-CSP-LMC-REQ-2279-02-00	**CSP Element Log File size** The CSP Log File shall store, at minimum, the most recent 2000000 errors or warnings
	SKA1-CSP-LMC-REQ-2279-03-00	**CSP Element Log File Copy** CSP_LOW.LMC and CSP_Mid.LMC shall provide interfaces for TM to obtain a copy of the current CSP Log File, in accordance with ICS CSP to TM.
	SKA1-CSP-LMC-REQ-2279-04-00	**CSP Element Log File Remote Access** CSP_LOW.LMC and CSP_Mid.LMC shall provide interfaces for TM to read and search the content of the current CSP Log File, in accordance with ICS CSP to TM.
LFAA.LMC	LMC-REQ 4040	**Standard logging** Use standard format, content and logging level as defined in “LMC Interface Guidelines”
	LMC-REQ 4050	**Remote logging** Report log messages to TM
	LMC-REQ 4060	**Control logging** Make provision for TM to control logging, including destination for logging messages and logging level
	LMC-0035	**Logging** Each component should generate appropriate logs at varying detail levels
	LMC-2055	**Application Logging** Functional application should be able to communicate with the system logger
	LMC-6110	**Logging Database** Log storage for future analysis and reporting
SDP.LMC		
SAT.LMC		

4.2 SKA Logging Architecture

4.2.1 LMC Logging Architecture

Describe here the models/patterns proposed for logging inside LMCs (local logging)

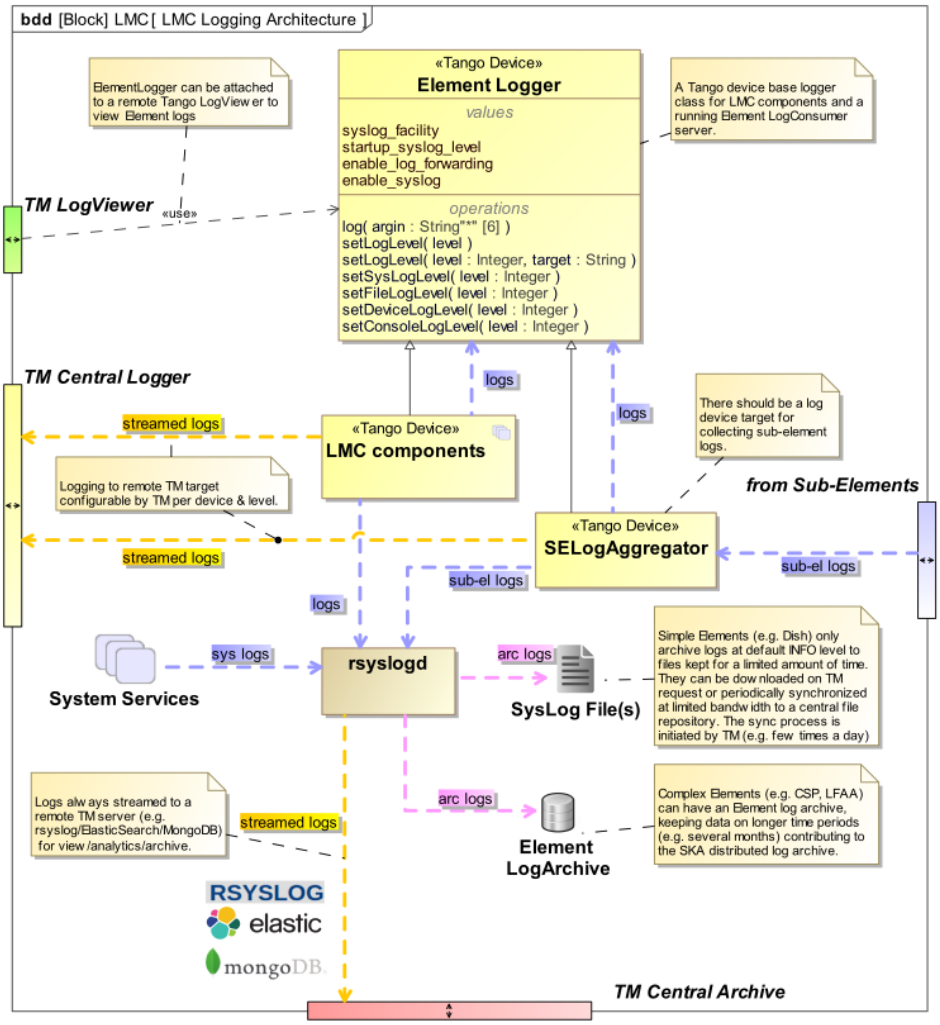

Fig. 4.2.1: *Schematic view of the proposed LMC Logging architecture*

A schema of the LMC logging architecture to be supported by all Elements is presented in Fig. 4.2.1, reflecting the following design pattern:

- Each LMC Tango device will always log to **2 log targets**:
    1. A local syslog server for log streaming and archiving purposes. syslog will be configured to archive incoming logs to local files and/or to a data storage backend and to forward them to a central location, e.g. a remote syslog server or data engine/store (see also Section 4.2.2 and discussion below).
    2. An *ElementLogger* device, implementing the Tango LogConsumer interface (see section 3.1.1)

and allowing to receive logs from any Tango device present in the LMC and in the managed sub-elements. The latter case is relative to some Elements only, for example the Dish, in which an aggregator device (named *SELogAggregator* in Fig. 4.2.1) collects all logs from Dish sub-elements (Single-Pixel Feed, Receiver and Dish Structure) via the LMC-Sub-Element interface. It has to be noted that complex Elements, like CSP or LFAA, could have indeed a hierarchy of Logger devices to distribute and balance the logging load. By ElementLogger we are therefore referring here to the top level LogConsumer (see also the Glossary section).
The *ElementLogger* enables TM to run instances of the Tango LogViewer with this unique endpoint device allowing to view logs generated in the Element for drill-down:

`$ logviewer $TANGO_HOST/[ElementLoggerName]` (TANGO_HOST set to the desired facility)

This has the advantage that TM does not need to go through the entire LMC device hierarchy to append log sources because all information is already aggregated in a single device. The ElementLogger device name is known to TM through naming convention and as an attribute exported on the LMC interface.

3. A CentralLogger device will be implemented by TM as the telescope LogConsumer with a central LogViewer. The ElementLoggers will forward device logs, with a default of ERROR and higher (TBD), or as configured by TM per device, to the CentralLogger for centralised log viewing across multiple Elements. The CentralLogger will archive these selected logs centrally. This is independent of the Element log archive and other mechanisms of sharing log archives between elements (like syslog forwarding or database sharding.)

Log levels are configurable by TM per device (e.g. on all or only selected devices) and per target (e.g. independently of each other) via commands provided on the LMC interface. The LMC in turn will set the log-level on the specified device direction through the TLS.

- By default, the INFO log level shall be specified for both targets in LMC devices.
- In addition to that targets, TM can also directly add/remove remote logging device targets on all/selected LMC devices at desired levels via commands provided on the LMC interface. Logs can therefore be directly streamed to TM from low-level devices. This will enable TM to build remote *CentralLogger*(s) devices supporting Tango LogViewer instances and view logs coming from different Tango facilities. Adding log sources belonging to different facilities is infact not supported at present by the Tango LogViewer.

The described model was evaluated against other possible architectures, for example a model in which archiving and reporting is fully demanded to a unique aggregator device. We note here the following aspects:

- Compared to the alternative model, in which there is a single overall point-of-failure for both log reporting and archiving, there are some advantages in having logs streamed and archived from each single device.
  If the *ElementLogger* goes down for whatever reason, logs from the other devices are still generated, reported and archived (if these devices are alive) independently from the *ElementLogger*. For viewing scopes TM can still access low-level devices, add a remote target directly to them being able to visualize logs, even in the absence of the *ElementLogger*. In the alternative model logs will be lost (not archived nor reported to TM) during the aggregator device downtime.
- The desired features, particularly the ability to log to syslog, have to be conveniently implemented in a *ElementLogger* base device inherited by all Element devices. This device has to be implemented in the

three programming languages (C++, Java, python) to support all LMCs implementations. This could represent a possible drawback in principle. However, as discussed in Section 3.3 and experimented during the prototyping, there are several builtin libraries available to considerably reduce the involved effort and all LMCs will benefit from the common base classes.

Policies for log streaming and archiving and the configuration strategy require an additional discussion, also going beyond the present document, reported in the following sections.

4.2.1.1 Log streaming at the Element

Log streaming from devices in the Element hierarchy to the ElementLogger for viewing purpose is by default set at INFO level but it could be eventually limited only to high-priority logs (e.g. WARN/ERROR/FATAL) in case of network bandwidth limitations. An INFO level is currently set in Meerkat telescopes and no issues have been reported so far with this configuration. A DEBUG level is configurable from the LMC interface but discouraged for normal operations.
Log streaming is also used for archiving purposes via syslog by "small" Elements not supporting local and persistent storage. In those cases the default log threshold is set at INFO level.

In general setting a log message level (particularly when promoting a DEBUG message to INFO) has to be therefore carefully evaluated by the Elements on a device basis.

4.2.1.2 Log archiving at the Element

Log archiving is performed via rsyslog forwarding macros to different backends (files, DB, remote syslog).
For simple LMCs, e.g. Dish, in which both computing and storage resources are limited by design and discouraged by archiving requirements, a temporary file-based archive will be present, that is logs will be archived locally in syslog files at INFO level and kept for a time period of few days at most. LMC makes provision for commands enabling download of log files to a desired location. Furthermore, if needed, a synchronization/backup process initiated by TM (and enabled by LMCs) could be designed to transfer log files to a central repository on a daily basis at a limited limited transfer speed (TBD).
Logs are also streamed to a central TM location for permanent archiving in a search engine, like Elastic, or in DB like MongoDB. This can occur with two different forward chains, supported by builtin rsyslog output modules:

- Element rsyslog forwarding to the remote TM rsyslog and the latter forwarding to MongoDB or Logstash/Elastic.
- Element rsyslog forwarding directly to remote MongoDB or Logstash/Elastic

The first approach, although implying more transfer stages, gives enough freedom and flexibility to both LMC and TM to explore suitable data storage solutions before a refined archiving strategy for SKA will be available.

For complex LMCs, e.g. CSP or LFAA, in which larger resources are allocated it would be instead desirable to have a dedicated long-persistence archive (e.g. month or one year) inside the Element, acting as a database shard for the central TM log archive. A distributed logging archive would be in this way realized with support for downloading DB dumps on request provided by LMCs. This option, advanced by TM.LMC and CSP, is however yet to be investigated in detail with interested Elements and TM.

4.2.1.1 Log streaming from Element to Central and Central Log Archiving at TM

Log streaming from *ElementLogger* to *CentralLogger* for centralised viewing of logs across all Elements, is by

default set at ERROR level but TM can update that by setting log level per device via the Element LMC.
The *CentralLogger* will archive these selected logs centrally. This is independent of the Element log archive and other mechanisms of sharing log archives between elements (like syslog forwarding or database sharding.)

4.2.1.3 Logging Configuration

Configuration of remote logging levels and targets is performed by TM by invoking commands provided by LMC in the interface device. These commands (the list is not exhaustive) are listed in Table 4.2.1 in pure Tango notation (e.g. no json format for input/output arguments). Command argument definition could change once a common strategy will be available.
It is expected that the implementation of required configuration commands is straightforward for LMC. This is infact easily achieved by combining builtin commands provided by the Tango Core (e.g. dserver admin and Group commands) and custom commands implemented in the *ElementLogger* device class (e.g. *setSysLogLevel*, *setDeviceLogLevel*, etc.) inherited by all LMC devices.
It has to be noted that in principle TM can directly access each single LMC device and invoke log configuration commands on them or implementing group commands to invoke collective log configuration actions. This is not prevented in the presented schema. Nevertheless we believe that having this collective commands already defined by LMC in the interface can ease TM considerably.

Table 4.2.1: *List of logging configuration commands*

Command	**Input Arguments**		**Output Arguments**		**Description**
	Type	***Description***	***Type***	***Description***	
SetLMCLogLevel	DevLong	The log level to be applied in Tango notation: 0=OFF 1=FATAL 2=ERROR 3=WARNING 4=INFO 5=DEBUG	DevVarLongStringArray	Long arg [0]: ack String arg [0]: err/info	Set the internal log level of Element LMC, e.g. the level of log reporting of each LMC device to the ElementLogger. By default this is set to INFO.
GetLMCLogLevel	DevVoid	-	DevVarLongStringArray	Long arg [0]: ack [1]: log level String arg [0]: err/info	Get the internal log level of Element LMC, e.g. the level of log reporting of each LMC device to the ElementLogger. By default this is set to INFO.
SetLMCArchivingLevel	DevLong	The log level to be applied in Tango notation: 0=OFF 1=FATAL 2=ERROR 3=WARNING 4=INFO 5=DEBUG	DevVarLongStringArray	Long arg [0]: ack String arg [0]: err/info	Set the archiving log level of Element LMC, e.g. the level of log reporting of each LMC device to syslog server. By default this is set to INFO.
GetLMCArchivingLevel	DevVoid	-	DevVarLongStringArray	Long arg [0]: ack	Get the archiving log level of Element LMC,

				[1]: log level String arg [0]: err/info	e.g. the level of log reporting of each LMC device to syslog server. By default this is set to INFO.
AddCentralLogger RemoveCentralLogger	DevVarLongStringArray	Long arg [0]: Log level String arg [0]: CentralLogger device (FQDN)	DevVarLongStringArray	Long arg [0]: ack String arg [0]: err/info	Add a central logger to each LMC device at the desired level.
SetLogLevel	DevVarLongStringArray	Long arg [0]: Log level String arg [0]: Device name [1]: Log target	DevVarLongStringArray	Long arg [0]: ack String arg [0]: err/info	Command for fine-tuning of logging level per LMC device and per target.
GetLogLevel	DevVarStringArray	[0]: Device name [1]: Log target	DevVarLongStringArray	Long arg [0]: ack [1]: log level String arg [0]: err/info	Command for retrieving the logging level per LMC device and per target.

On the other hand, configuration of syslog (remote server, log filters and templates, etc.) typically resides on configuration files (e.g. /etc/rsyslog.conf, /etc/rsyslog.d/...) not intended to be dynamically modified at runtime. A standardized file-based configuration shall be therefore set up by LMCs and TM jointly and kept as default for normal operation. The established configuration could be however changed during maintenance periods, e.g. using configuration automation tools such as Puppet, Chef, Ansible. This goes however beyond the scope of the document.

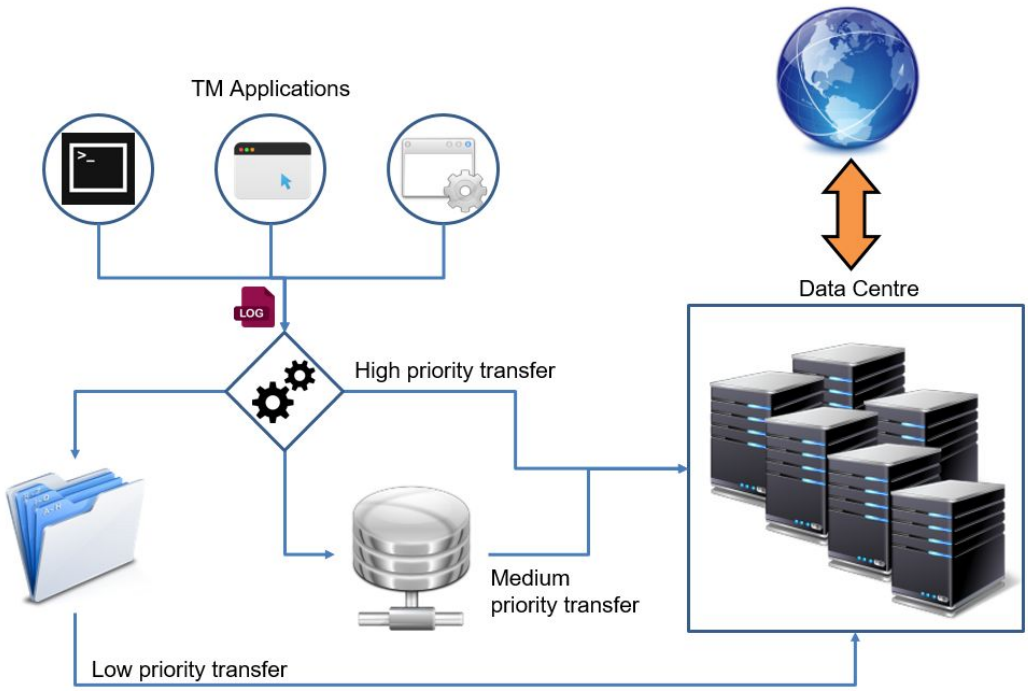

Fig. 4.3.1: Schema of TM Logging Architecture

4.2.2 TM Logging Architecture

Describe here the models/patterns proposed for logging inside TM (element logging for TM) and for reporting and archiving LMC logs from Elements centrally.

A unique logging service will be designed for TM applications (TM.OBSMGT, TM.TELMGT, TM.LMC, TM.LINFRA) and SKA applications (all LMC Elements) and should be based on proven best practises.
In Figure 4.3.1, it is shown the data flow of log informations from the client applications (basically all SKA Applications) to the server data centre in a possible architecture. The data center is a hierarchy and potentially every tango domain should have a specific database cluster to collect log messages and increase performance.
The central point of the figure and the architecture is a service (generally called Log Forwarder) located near the applications) that give the possibility to forward the message to the central database cluster.
The forward mechanism has a priority level for every message:

- Lower priority: the log messages (files) are copied from a local folder to the main data center without any decrease of network performance;
- Medium priority: the log messages (files) are directly written in a network path or in a cloud folder so that it can be synchronized with the main data center;
- High priority; the log messages (files) are directly sent through the tango logging system because there a need for the maintainer or the developer to look at them as soon as possible.

The proposed architecture is a best practise and basically composed by three main entities: the server (or data centre), the client and the forwarder. The data centre is the entity responsible for collecting and organizing the message from every applications; the client helps the applications to compose the log message (for instance with macro or template) and can be configured remotely to use a forwarder which is the entity responsible for the transfer of the informations to the server.
Another important aspect is the possibility for the users of every SKA applications to retrieve the log files which they are interested in: this possibility should be realized by a web application usable by every user which can directly query the data centre in order to retrieve entire log file or a collection of log files or a specific result of a query.
The growth of event data should be controlled in order to avoid the storage of unused informations and in order to maintain an enough amount of data persistent without the risk of data flooding. There are different possibilities and one of the most used one is to have a fixed size for data and drop all the messages which exceed that data. Other possibilities are with the use of some noSql technology like for instance MongoDB which allow the operator to have multiple db instances and directly drop an instance when needed.

4.3 Log format

Describe here the logging format to be adopted for SKA.

4.3.1 Tango log format (CONSOLE/DEVICE/FILE/VIEWER)

LMC Tango devices shall adhere to Tango Guidelines [RD15] when logging to console/device/file/viewer targets. The log message shall be formatted as follows:

<PREFIX> - <MSG TEXT>

where <PREFIX> is given by: *<CLASS_NAME>::<FUNCTION>()*. This is the default logging format present in the device code generated with Pogo.

Log messages generated in functions/classes within the same device class namespace shall be sent “in the device name and follow the same format prescription given above. The Tango manual specifies how to log in the name of a device (e.g. see section 6.3.3.2 of [RD2] - *C++ logging in the name of a device*).

4.3.2 Tango log format (SYSLOG)

The syslog message header and content field shall be set by the application generating the log message. This means that Tango devices logging to syslog shall set the syslog message fields according to the prescriptions given in Table 4.3.1.

Table 4.3.1: *Log format prescriptions to be adopted when filling syslog message in Tango devices*

Field	Priority	Prescription
FACILITY	Mandatory	Set to one of the values: *local0 - local7*
SEVERITY	Mandatory	Set according to the mapping reported in Table 5.1.3.2
HOSTNAME	Optional, but desirable	Set to the host information where the Tango device is running, with this order of preference: 1. FQDN 2. Static IP address 3. Hostname 4. Dynamic IP address 5. the NILVALUE
TIMESTAMP	Mandatory	Set in conformance with ISO 8601 or RFC-3339 with sub-second precision and in universal time (TBC)
TAG	Optional (see prescription)	It could be used to specify the device name (if smaller than 32 char). If so, the field is mandatory to be set.
APP-NAME (RFC5424)	Optional, but desirable	It could be used to specify the device name or the device server name.
PROCID (RFC5424)	Optional	Set to the pid of the running device server.
CONTENT	Mandatory	It shall not be an empty string. It shall conform to the Tango style notation described in Tango guidelines [RD15] and in Section 4.3.1: *<CLASS_NAME>::<FUNCTION>() - <MSG TEXT>* *Example: MyClass::MyFunction() - A log message*

4.4 Adopted technologies

Report here the technology adopted and the results of the evaluation (if performed)

TBD

5 Logger Prototypes

Report here the prototyping activities performed

We discuss in this section the prototyping activities (in C++ and Java) carried out on the basis of the logging architecture model and technological solution explored.

5.1 C++ Element Logger on ELK Stack

We developed a simple Logger device prototype in C++. Its class diagram is reported in Fig. 5.1.1.

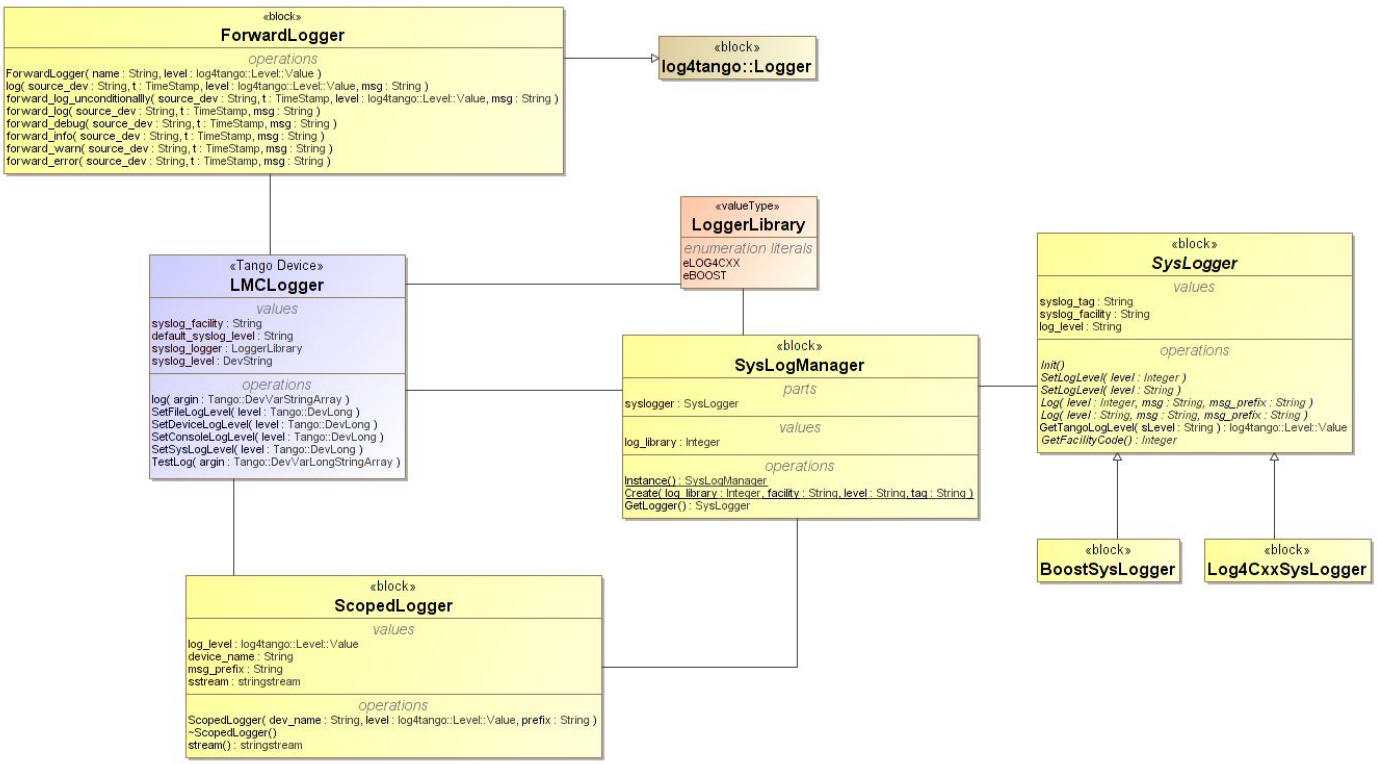

Fig. 5.1.1: Architecture schema of LMCLogger C++ device prototype

The device allows to log to all Tango log targets (provided that Tango was built with the option) and to syslog at different levels, configurable via the provided commands. Some helper logging macros are provided:

- LOG(level,"msg")
- INFO_LOG("msg")
- ERROR_LOG("msg")
- WARN_LOG("msg")
- DEBUG_LOG("msg")

A predefined log prefix ("<CLASS_NAME>::<FUNCTION_NAME>() - ") is automatically added to the message in conformance with Tango guidelines.
The defined device properties enables configuration of default startup syslog level, facility and syslog logger library (BOOST, Log4Cxx at present).
The device implements also the Tango LogConsumer interface, enabling log forwarding features. Tango logging API does not allow to specify the original logging source and timestamp in forwarded log messages. This limitation can be overcome either by slightly modifying the Tango logging core components (e.g. adding the

desired methods) or introducing a *ForwardLogger* class inherited by the log4tango::Logger and providing the desired functionalities, for example:

```
void ForwardLogger::fw_log_unconditionally(std::string source_device, Level::Value level, const
std::string& message,Timestamp& original_time)
{
    #ifdef LOG4TANGO_HAS_NDC
    LoggingEvent event(source_device, message, NDC::get(), level,&original_time);
    #else
    LoggingEvent event(source_device, message, level,&original_time);
   #endif
  call_appenders(event);
}
```

The latter approach has the advantage that no modification of Tango Core is required. On the other hand, one has to provide additional methods and attributes to allow configuration of the forwarding level (not shown in the diagram). In the former approach configuration is realized by using standard dserver commands already provided by Tango.

For demonstration purpose we setup a local rsyslog server (i.e. the LMC) in which we run an instance of the LMCLogger device server named *dshlmc/lmclogger/id1* configured to log to syslog using facility *local6*. The following (intentionally verbose) rsyslog template (here named *CustomFormat*) was assumed:

```
template(name="CustomFormat" type="list") {
  constant(value="")
    property(name="timereported" dateFormat="rfc3339" date.inUTC="on")
  constant(value=" ")
  constant(value="[fromhost: ")
    property(name="fromhost") constant(value="] ")
  constant(value="[hostname: ")
    property(name="hostname") constant(value="] ")
  constant(value="[severity: ")
    property(name="syslogseverity-text" caseconversion="upper")
  constant(value="] ")
  constant(value="[facility: ")
    property(name="syslogfacility-text" caseconversion="lower")
  constant(value="] ")
  constant(value="[app-name: ")
    property(name="app-name" caseconversion="lower")
  constant(value="] ")
  constant(value="[pri: ")
    property(name="pri") constant(value="] ")
  constant(value="[tag: ")
    property(name="syslogtag") constant(value="] ")
  constant(value="[struct: ")
    property(name="structured-data")
  constant(value="] ")
  constant(value="[msgid: ")
    property(name="msgid") constant(value="] ")
  constant(value="[msgcontent: ")
    property(name="msg")
  constant(value="]")
```

```
  constant(value="\n")
}
local6.*    /var/log/tango.log;CustomFormat
local6.warn @XXXX:514 ## Forward only Tango logs
```

All logs generated from LMC Tango devices will be therefore archived locally in /var/log/tango.log using the CustomFormat format and only logs with severity higher than “warn” are forwarded to a remote server XXX listening on UDP port 514.

We set up also a remote host (i.e. at TM level) with rsyslog, logstash and elasticsearch server running and configured as follows:

```
#==================================
#==     Remote rsyslog configuration
#==================================
## Enable log listening on udp & tcp
$ModLoad imudp
$UDPServerRun 514
$ModLoad imtcp
$InputTCPServerRun 514

## Set Tango log format and file (same as before)
$template     CustomFormat,"%timereported:::date-rfc3339%     [fromhost:     %FROMHOST%]     [severity:
%syslogseverity-text:::UPPERCASE%]  [app-name:  %app-name%]  [pri:  %pri%]  [tag:  %syslogtag%]  [struct:
%structured-data%] [msgid: %msgid%] [msgcontent: %msg%]\n"
local6.* /var/log/tango.log;CustomFormat

### Configuration for logstash forwarding
template(name="json-template"
  type="list") {
      constant(value="{")
      constant(value="\"@timestamp\":\"")     property(name="timereported" dateFormat="rfc3339")
      constant(value="\",\"@version\":\"1")
      constant(value="\",\"message\":\"")     property(name="msg" format="json")
      constant(value="\",\"sysloghost\":\"")  property(name="hostname")
      constant(value="\",\"sourcehost\":\"")  property(name="fromhost")
     constant(value="\",\"severity\":\"")    property(name="syslogseverity-text" caseConversion="upper")
     constant(value="\",\"facility\":\"")    property(name="syslogfacility-text")
     constant(value="\",\"tag\":\"")         property(name="syslogtag")
     constant(value="\",\"programname\":\"") property(name="programname")
     constant(value="\",\"app-name\":\"")     property(name="app-name")
     constant(value="\",\"procid\":\"")      property(name="procid")
     constant(value="\"}\n")
}
## Forward to logstash listening on port 10514
local6.* @localhost:10514;json-template

# ========================
#==   Logstash config            ==
#========================
input {
   udp {
      host => "localhost"
```

```
      port => 10514
      codec => "json"
      type => "rsyslog"
   }
}

output {
   if [type] == "rsyslog" {
      elasticsearch {
         hosts => [ "localhost:9200" ]
      }
   }
}
```

Logs generated by LMC are therefore received by the remote rsyslog server, forwarded using a json encoding to the logstash server (listening on port 10514) and then to the elasticsearch engine where they can be searched or viewed, e.g. using Kibana. Logstash and elasticsearch are in this case running on the same rsyslog remote host. Rsyslog provides also an output module (omelasticsearch) allowing to forward logs directly to elastic without the intermediate steps (remote rsyslog + logstash) worth to be tested.

To test the setup we generated a sample FATAL log message from the LMCLogger Tango device (named *dshlmc/lmclogger/id1)*. Below the log displayed by the local rsyslog using Log4Cxx logging library:

```
2016-05-24T19:22:35.374869+02:00 [fromhost: localhost] [severity: CRIT] [app-name: dshlmc] [pri: 179] [tag: dshlmc/lmclogger/id1] [struct: -] [msgid: -] [msgcontent:  LMCLogger::test_log() - A fatal message]
```

Below the log displayed by rsyslog using Boost.log logging library:

```
2016-05-24T19:50:34.911663+02:00 [fromhost: riggi-XXXXXX] [severity: CRIT] [app-name: dshlmc] [pri: 182] [tag: dshlmc/lmclogger/id1:] [struct: -] [msgid: -] [msgcontent:  LMCLogger::test_log() - A fatal message]
```

It seems that the host information (riggi-XXXX) cannot be properly set by Log4Cxx. Also, some loggers add a ":" at the end of tag (see syslog RFC 5.3 Originating Process Information).

On the remote host we retrieved the generated log in the elastic engine:

```
$ curl -XGET
    'http://localhost:9200/_all/_search?q=tag:dshlmc\/lmclogger\/id1%20AND%20severity:CRIT&pretty'

{
  "took" : 135,
  "timed_out" : false,
  "_shards" : {
    "total" : 6,
    "successful" : 6,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 6.898387,
    "hits" : [ {
      "_index" : "logstash-2016.05.26",
```

```
      "_type" : "rsyslog",
      "_id" : "AVTuH5s9MaS4UqK5Htjt",
      "_score" : 6.898387,
      "_source" : {
        "@timestamp" : "2016-05-26T17:32:08.000Z",
        "@version" : "1",
        "message" : " LMCLogger::test_log() - A fatal message",
        "sysloghost" : "riggi-XXXXX",
        "sourcehost" : "XXXXX",
        "severity" : "CRIT",
        "facility" : "local6",
        "tag" : "dshlmc/lmclogger/id1:",
        "programname" : "dshlmc",
        "app-name" : "dshlmc",
        "procid" : "-",
        "type" : "rsyslog",
        "host" : "127.0.0.1"
      }
    } ]
  }
}
```


The generated log can be viewed in Kibana (see 3.5.1).

5.2 TM Logging System

TBD

6. Summary

Add summary comments

TBD

In this document we investigated several aspects related to SKA logging, from logging architecture at the Element to suitable strategies for log reporting to TM and archiving. Further investigations are still required for some crucial issue, like log archiving, as we discussed in the document. We summarize here the main outcomes of the performed analysis and the guidelines to be followed by SKA LMC Elements:

Element LMC

- Provide an *ElementLogger* device in the control system architecture implementing the LogConsumer interface
- Provide a local rsyslog server in the system with standardized configuration enabling file and remote rsyslog backends
- Specify 2 logging targets at default INFO level in each LMC Tango device: *ElementLogger* + rsyslog
- Provide support for adding/removing a CentralLogger device as target to one/all Element devices
- Provide commands to enable logging configuration as described in Table 4.2.1
- Log device messages in the format defined in Section 4.3.1 and 4.3.2

TM

- Provide a remote rsyslog server for receiving logs from local LMC rsyslog servers

- Provide CentralLogger device(s) in the control system architecture implementing the LogConsumer interface

Appendix

Define SKA design pattern: (TBC)

- **TBC**: Lize added: The term **ElementLogger (**elsewhere called LMCLogger in this doc at this point in time) is used to refer to the top-level LogConsumer within the Element. The ElementLogger will be the default LogConsumer for all devices in the Element hierarchy, and also provide LogViewing.
  [TBC which implementation for remote logging to CentralLogger. Options are:
  1) ElementLogger does remote logging to CentralLogger on behalf of devices in the Element hierarchy
  2) CentralLogger target is added/remove on each device as required in which case device directly sends log to CentralLogger and not via ElementLogger
  3) other??]
- **TBC**: Lize added: The term **CentralLogger** (elsewhere called Central LogConsumer at this point in time) is used to refer to the central LogConsumer at TM that collates logs across all Tango facilities in the telescope. The CentralLogger will be the default remote LogConsumer for all ElementLoggers [**TBC**, unless we decide that lower level devices will directly log to CentralLogger when requested by TM]
- **TBC**: Lize added: The TM will manage remote logging of devices in the Element hierarchy through the **ElementLMC**. The ElementLMC will distribute the requests for remote logging and the remote log level per device as needed to the ElementLogger which in turn will add and remove logging targets and set logging levels on devices in the Element hierarchy as required. [**TBC** - is this the pattern we want? Alternatives are:
  1) TM manages remote logging through ElementLogger
  2) TM manages remote logging through ElementLMC, which in turns manages it through ElementLogger
  3) TM manages remote logging through ElementLMC, which manages remote logging targets directly on lower level devices
  4) [Not desirable but possible] TM manages remote logging directly on lower level devices in the Element hierarchy by adding/removing logging target for CentralLogger.
  5) other???
- SKA Tango devices will log to syslog at default INFO level and also have a local LMCLogger device (LogConsumer) logging target at default INFO level.
  How will Tango devices "know" the Element LMC Logger? Through naming?
- Each Element will have a SKA LMCLogger (LogConsumer) to support LogViewer and manage/forward logs remotely at a configurable level
- Element LMC will be used to configure remote central logging and remote logging level for lower level Tango devices. The devices will have the ElementLogger as default logging target and the ElementLoggers will have the CentralLogger as default logging target. The remote logging will be forwarded from the ElementLogger at a default level of ERROR, or as configured differently per device by TM. To support central viewing of logs from multiple Elements the log message format should contain the device name.

- Do we want to specify the default remote logging level & behaviour - is it OFF or is it WARN or ERROR? If it is not off how will the devices "know" the central LogConsumer, through naming?
- Element LMC will instruct lower level Tango devices or Element LogConsumer when getting commanded to enable/disable central logging for a device
- TBD Where/should Element logs will be archived locally in the ELement, if at all, or just in syslog files, or sync/backup the files somewhere - locally or centrally
- For central log viewing, is it possible for TM to manage navigation or open the Element LogViewer (or connect a LogViewre to the Element LogConsumer)? Tango prototypers might be able to answer.
- Should include here the other critical points from the discussions above: e.g. minimum length of time to keep Element log files, mechanism to sync Element log files to a central archive (if necessary) etc