

LogM System Architecture Description

Log Management Framework

DESCRIPTION

Copyright

© Ericsson AB 2017–2019. All rights reserved. No part of this document may be reproduced in any form without the written permission of the copyright owner.

Disclaimer

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

All trademarks mentioned herein are the property of their respective owners. These are shown in the document Trademark Information.



Contents

1	Introduction	1
2	Overview	3
3	Architectural Goals and Constraints	5
3.1	Goals	5
3.2	Constraints	6
4	Logical View	7
4.1	Detailed Overview of Log Framework	7
4.2	Interfaces	9
5	Use Case View	11
5.1	Log Stream Registration	11
5.2	Log Stream Unregistration	12
5.3	Change in Severity Filter	14
5.4	Export Log Stream	15
6	Process View	17
6.1	Core MW LogM Users and Groups	17
6.2	Core MW LogM Processes	17
7	ECIM Security Roles and Rules	19
7.1	System Administrator Role	19
7.2	System Security Administrator Role	19
7.3	System Read Only Role	19
8	Deployment View	21
8.1	Redundancy	21
8.2	Failover and Switchover Operations	21
8.3	Linux System Solutions	21
8.4	Alarms	21
9	Implementation View	23
9.1	Overview	23
9.2	Logging and Trace	24
10	Data View	25



11	Size and Performance	27
12	Quality	29
13	Third-Party Components	31



1 Introduction

This document describes the functionality and software architecture of the Core Middleware (Core MW) Log Management (LogM) Framework.

This document is intended to be a central reference for system managers, system designers, developers, testers, project managers, and product managers who need to understand the structure and organization of Core MW LogM.





2 Overview

Core MW LogM provides the centralized functionality to manage log streams. In the Component Based Architecture (CBA) environment, LogM is integrated with the Core MW and Linux® Distribution Extensions (LDE) products.

The following functions are included in Core MW LogM:

- Core MW LogM facilitates the registration of log streams in the CBA environment.
- Core MW LogM supports the centralized management by the Northbound Interface (NBI) of registered log streams, including setting severity filter, performing manual export of logs, and configuring automatic streaming of log entries towards a log server.

Applications that are log stream owners and create log streams in CBA may register their inventory of log streams with Core MW LogM so that they can be managed by the NBI.

For more information about log stream registration details and requirements, refer to [CoreMW LogM Description](#).





3 Architectural Goals and Constraints

3.1 Goals

3.1.1 Standardized Interfaces

Core MW LogM provides an implementation of the Ericsson Common Information Model (ECIM) LogM 2.0 interface. Unique Operations Support System (OSS) adaptations are not required.

Core MW LogM offers the Log Management Application Programming Interface (API) to enable each log framework owner to create a Log Controller component to be used in the CBA log framework.

For more information about interfaces used and offered by Core MW LogM, see Section 4.2 Interfaces on page 9.

3.1.2 Scalability

Scaling down the system is important to have low-entry cost for small systems. Scaling up the system is important to be able to provide a solution matching workload requirements.

Core MW LogM does not put any restrictions on the number of nodes or the time required to complete log management operations.

3.1.3 Hardware Transparency

Core MW LogM can run on different Commercial Off-The-Shelf (COTS) hardware and Ericsson Blade System (EBS) hardware. Core MW LogM can be designed to support different processor hardware architectures.

3.1.4 Offline Installation and Instantiation

Core MW LogM can be installed and instantiated, either as an individual component, that is, on top of a running Linux cluster, or as part of a prebuilt offline disk image. The latter method means that all software, the Linux operating system, Core MW, Core MW LogM, and all other CBA components, already reside on a disk image ready to be instantiated in Virtual Network Function (VNF) infrastructure, like cloud environments.

When a disk image is instantiated the first time, Core MW bootstraps an instantiation campaign, involving all CBA components including Core MW LogM, on top of Core MW, and then executes this campaign so that the CBA components are instantiated.



3.2 Constraints

3.2.1 x86-64 Architecture

Core MW LogM only supports the Intel® x86-64 architecture.

3.2.2 Operating System

Core MW LogM is a Linux Standard Base (LSB) compliant product. Core MW LogM requires Linux as the target operating system.

3.2.3 Personal Data Requirement

Core MW LogM manages and configures registered log streams but has no knowledge of the contents. A component or application that is a log stream user is responsible for managing the data according to Ericsson personal data requirements.



4 Logical View

This section contains a functional overview of Core MW LogM functionality, including the interfaces Core MW LogM offers and uses.

Core MW LogM is a framework that enables applications or components to perform log management operations on registered log streams. This framework includes a shared library named LogM Library and the Core MW Log Controller binary.

The LogM Library delivers the CmwLogM 1.0.2 management model and implements the common and generic functionality used by all log controllers.

The Core MW Log Controller implements the SAFLOG environment-specific functionality, needed to perform the log framework operations on registered SAFLOG runtime streams.

4.1 Detailed Overview of Log Framework

The logical overview of the full log framework in CBA is shown in Figure 1.

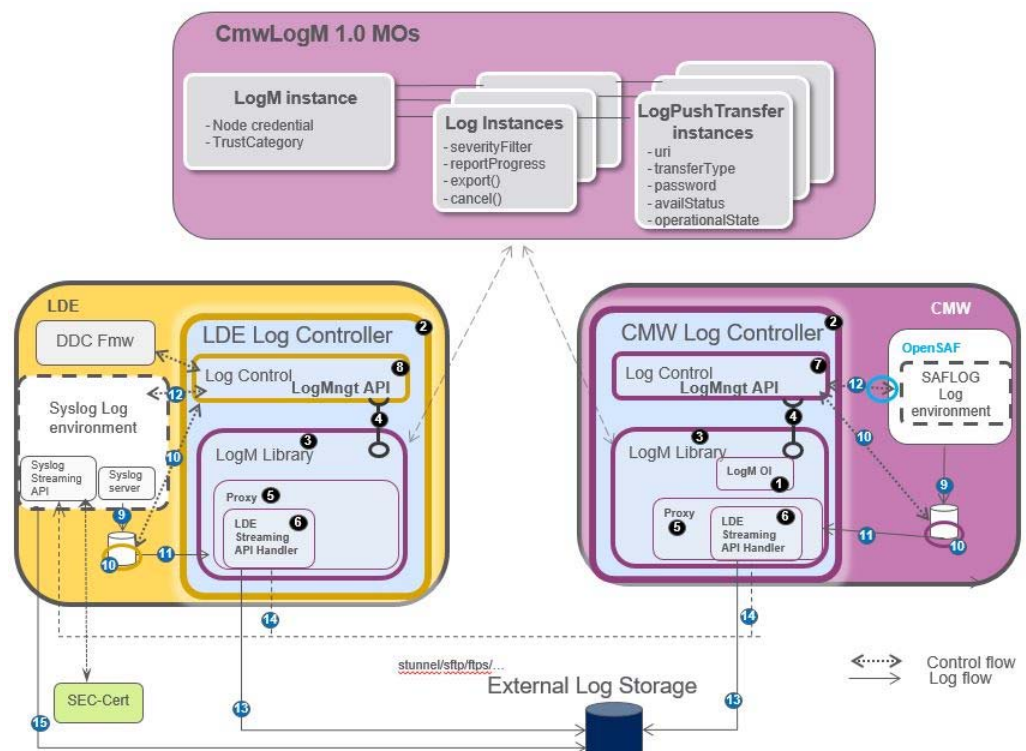


Figure 1 Log Framework in CBA Logical Overview

Description of the elements and actors in Figure 1:



1. Log Controllers – Run in the execution environment of each log framework owner. Core MW Log Controller is run 1+1 in the Availability Management Framework (AMF) environment and LDE Log Controller is run in 2N + NoRed redundancy.
 - Only one Log Controller instance runs per log framework owner.
 - Once an instance of the Log Controller identifies and registers a log stream belonging to the log framework owner, it then becomes responsible to apply the configurations done to LogM MOs for the registered log streams.
2. LogM Library – Core MW provides this entity. It implements a shared library of common functionalities (export/streaming management) and 2 APIs (LogMgmtController and LogMgmtStreaming API) to interact with the services in the log framework. It also implements common functionality for Information Model Management (IMM) needed for each service. The LogM Library runs in the process space of a Controller or Streaming Service instance.
3. LogMgmtController API – Simplified API provided by Core MW giving access to the functionalities of the LogM Library to a Log Controller service. For example, provides severity filter configuration, export requests, and enable or disable streaming requests to a Log Controller that has registered log streams with the Library.

Validates SEC parameters and provides them to abstracted LDE Streaming API, which is used towards the SEC CREDU API to retrieve the certificates needed to support streaming over Transport Layer Security (TLS).
4. LDE Streaming Service - AMF aware component using the shared LogM Library to provide the streaming backend. Validates SEC parameters and provides them to SEC CREDU API to retrieve the certificates needed to support streaming over Transport Layer Security (TLS). Consumes the streaming configuration for registered log streams to enable to automatic log streaming
5. Core MW Log Control – Control functions provided by Core MW to manage the SAFLOG Runtime log streams and SAFLOG configuration object in IMM. Implements the interaction with the SAFLOG log environment and interaction with the LogM Library by the LogMgmtController API.
6. LDE Log Control – Control functions provided by LDE to manage the syslog log streams. Implements the interaction with the syslog environment, and interaction with the LogM Library by the LogMgmtController API.

Note: Figure 1 shows the full Log Framework in CBA including elements implemented by LDE with SUSE™ Linux Enterprise Server (LDEwS), and only the elements highlighted in purple are implemented by Core MW LogM. This includes the CmwLogM model, the LogM Library, and the Core MW Log Controller.



4.2 Interfaces

4.2.1 Consumed Interfaces

The consumed interfaces are listed in Table 1.

Table 1 Consumed Interfaces

Product	Consumed Interface
Core MW	<ul style="list-style-type: none"> • IMM service • AMF service • LOG service • MDF service – Core MW LogM provides an MDF model consumer for LOGM_R1 model type. In addition, Core MW LogM itself delivers models consumed by the following MDF model type consumers: • COM_R1 • IMM_R3 • LOGM_R1
LDEwS	<ul style="list-style-type: none"> • SUGAR API • OpenSSL

4.2.2 Provided Interfaces

4.2.2.1 Log Management API

Core MW LogM provides both LogMgmtController API and LogMgmtStreaming API as a C header file.

The Log Management Controller API is a simplified API provided by Core MW LogM giving access to the functionalities of the LogM Library needed by Log Controllers. This API allows the Log Controller binaries to include the shared LogM Library containing the generic functionality of the log framework.

For more information, refer to [LogMgmtController API](#).

The Log Management Streaming API is a simplified API provided by Core MW LogM allowing a Log Streaming Service to access the generic functionalities of the LogM Library and to consume the streaming related configuration needed to support the automatic log streaming use case.



4.2.2.2 ECIM Interfaces

The implemented ECIM models are listed in Table 2.

Table 2 ECIM Models

Model Name	Model Version	Derived from Model	Compliance Information	Reference
CmwLogM	1.0.2	ECIM LOGM 2.0	Fully Compliant	Core MW ECIM Statement of Compliance

The LogM fragment of the ECIM model is used to configure and control Core MW LogM.

Applications use it to configure log streams and monitor the progress of ongoing operations.



5 Use Case View

The following Core MW LogM use cases are described in detail:

- Log stream registration
- Log stream unregistration
- Change in severity filter
- Export log stream

5.1 Log Stream Registration

The log stream registration sequence diagram is shown in Figure 2.

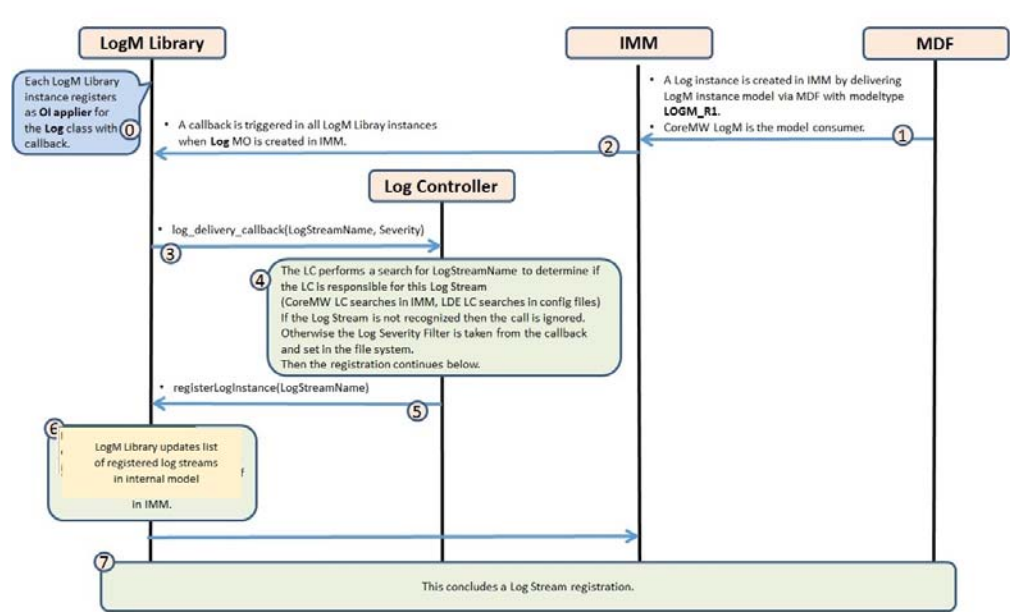


Figure 2 Log Stream Registration Sequence

Preconditions

- Core MW LogM is instantiated and running.
- Core MW LogM registers as Model Delivery Framework (MDF) model consumer for the LOGM_R1 model type.
- At startup, all LogM Library instances register themselves as IMM OI appliers for the Log class. This provides a mechanism for them to be notified when a Log instance is created in IMM.



Detailed Description

1. The LogM instance models with model type LOGM_R1 are queued by MDF when a CC is installed and delivered immediately if Core MW LogM (the model consumer) is installed. The MDF delivery creates Log MOs in IMM for the log streams in the instance models.
2. Each LogM Library instance is the OI (applier) for the Log class and receives a callback for each created MO.
3. The LogM Library instance requests the Log Controller to identify the log stream by a LogMgmtController API callback.
4. The Log Controller uses the log stream name and determines if it is responsible to handle this log stream and checks if it belongs to the local log environment. For example:
 - The Core MW Log Controller checks in IMM for SAFLOG Runtime streams in attempt to identify the log stream by the name.
 - The LDE Log Controller checks against known list of syslog stream names, which are provided by log stream owners.

If a Log Controller does not identify the log stream name, then nothing happens. Otherwise it applies the provided severity filter value for the successfully registered log stream.
5. If the Log Controller determines that it is responsible for this log stream, then it calls the LogMgmtController API function `registerLogInstance()`.
6. The LogM Library instance registers itself as the single OI for Log instances with IMM.
7. The registration is completed.

5.2 Log Stream Unregistration

The log stream unregistration sequence diagram is shown in Figure 3.

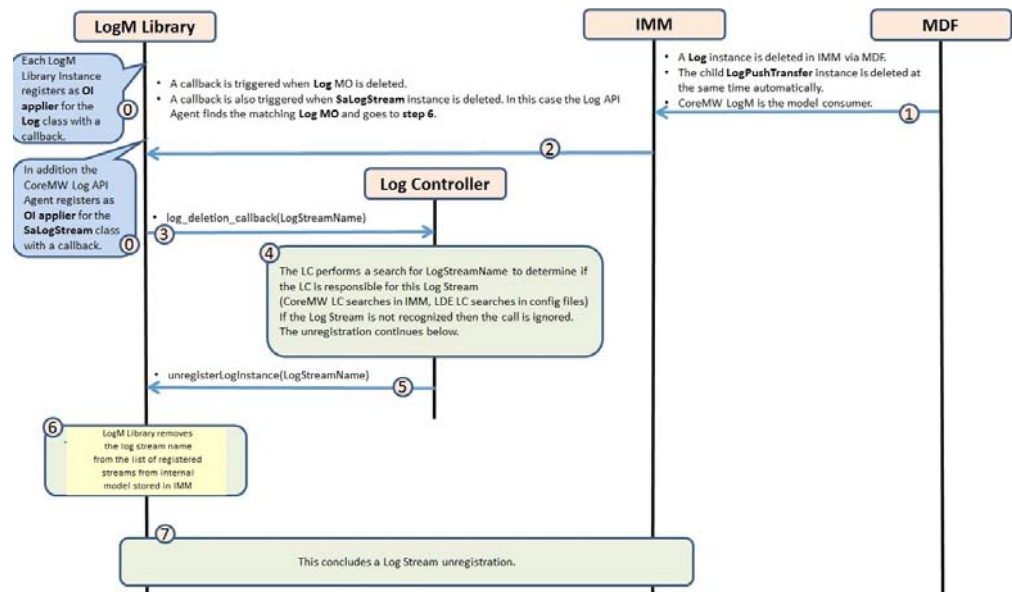


Figure 3 Log Stream Unregistration Sequence

Preconditions

- Core MW LogM is instantiated and log streams are already registered.

Detailed Description

1. The LogM instance models are deleted by MDF when a CC is uninstalled or upgraded. Core MW LogM is the model consumer of the LOGM_R1 model type. The MDF operation deletes the specified Log MOs in IMM for all log streams represented in the instance models.
2. Each LogM Library instance is OI (applier) for the Log class and receives a callback for each deleted MO.
3. The LogM Library notifies the Log Controller by using the LogMgmtController API callback `log_deletion_callback(LogStreamName)`.
4. The Log Controller performs a search to confirm that the stream was previously registered:
 - Core MW checks in IMM for the Runtime SAFLOG stream to identify the log stream name.
 - LDE checks against configuration files, which are provided by log stream owners. These configuration files contain log stream names.

If a Log Controller identifies the log stream from the Log MO, then it proceeds with Step 5. Otherwise it does nothing.

5. The Log Controller calls the LogMgmtController API function `unregisterLogInstance(LogStreamName)`.
6. The LogM Library instance unregisters itself as the main OI for this Log instance and removes the log stream details from the internal model.
7. The log stream unregistration is completed.

5.3 Change in Severity Filter

The change in severity filter sequence diagram is shown in Figure 4.

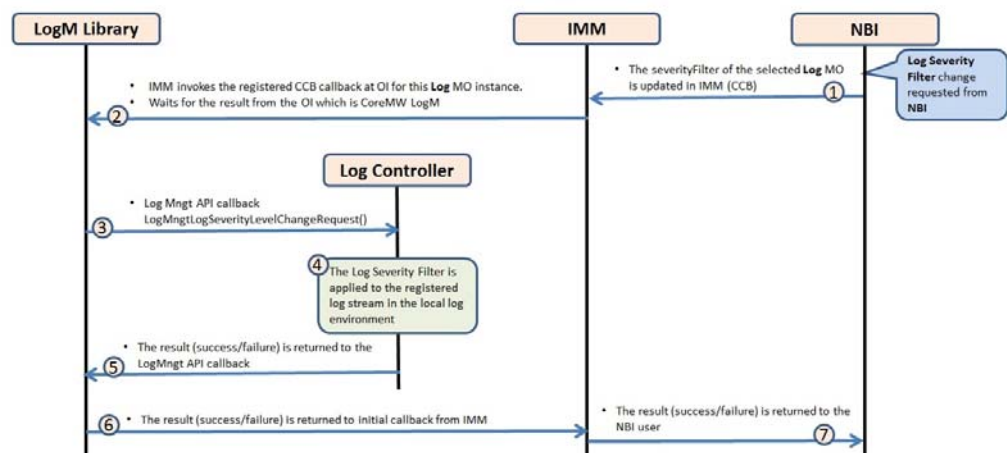


Figure 4 Change in Severity Filter Sequence

Preconditions

- Core MW LogM is instantiated and the log stream is successfully registered.

Detailed Description

1. The NBI user configures the `severityFilter` attribute of the Log MO.
2. The IMM service starts the callback function at OI (Core MW LogM Library instance), which needs to participate in the Configuration Change Bundle (CCB) to apply the configuration.
3. The LogM Library starts `LogMgmtControllerSeverityLevelChangeRequest()` callback in the appropriate Log Controller owning the registered log stream.
4. The Log Severity Filter is applied to the registered log stream in the local log environment by the Log Controller.
5. The result (success/failure) is returned to the LogMgmtController API callback.
6. The result is returned to IMM by the callback.
7. The result (success/failure) is provided to the NBI user.



5.4 Export Log Stream

The export log stream sequence diagram is shown in Figure 5.

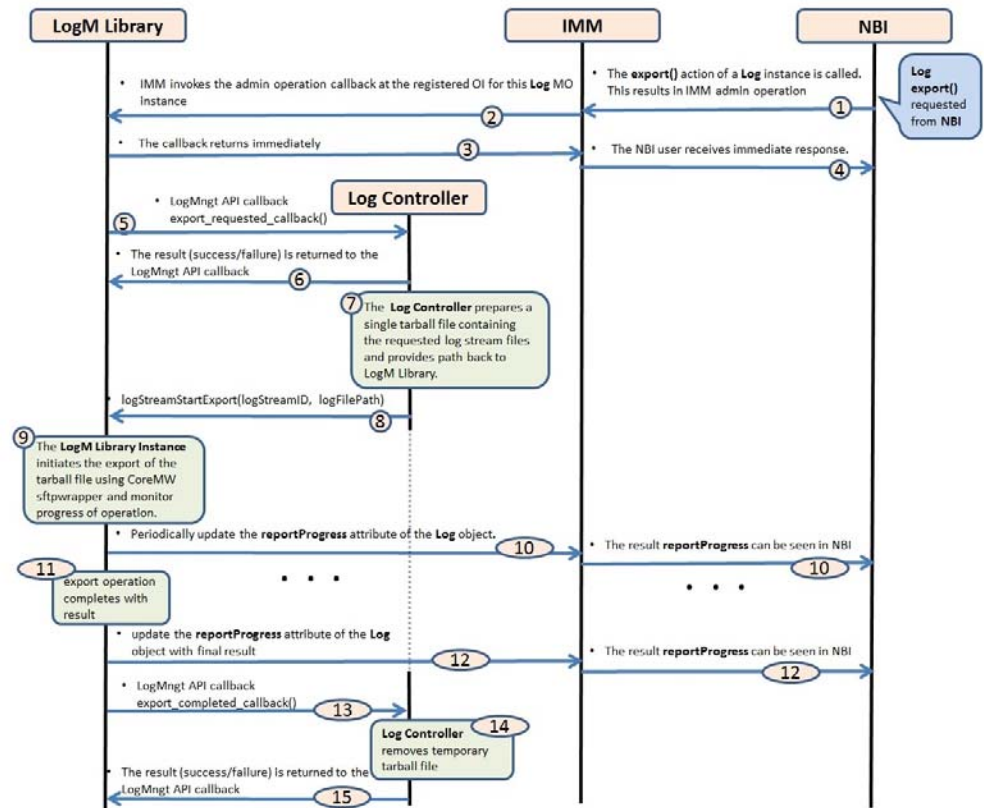


Figure 5 Export Log Stream Sequence

Preconditions

- Core MW LogM is instantiated and the Log stream is successfully registered.
- The log stream has been opened by the application and the corresponding log files exist on target.

Detailed Description

1. The NBI user navigates to the Log instance and starts the **export** action (with **uri**, **password**, and **transferType**).
2. IMM starts the registered admin operation callback at OI. It does not wait for the result.
3. The callback function returns immediately (it is asynchronous).
4. The NBI user receives immediate confirmation (for example, 'true' or 'ok').



5. The LogM Library calls the LogMgmtController API callback `export_requested_callback(LogStreamName)`.
6. The result (success/failure) is returned to the LogMgmtController API callback.
7. The Log Controller finds the log filenames on target corresponding to the registered log stream. The Log Controller prepares a temporary tarball file containing the log files to be used for export.
8. The Log Controller starts `logStreamStartExport(logStreamID, tarballFilePath)` over the LogMgmt API.
9. The LogM Library uses the Core MW sftpwrapper to perform the export of the tarball over SFTP by creating a child process.
10. The LogM Library periodically updates the `reportProgress` attribute of the corresponding Log instance during the export operation.

The progress report can be seen in the NBI.

11. The export operation completes with a result provided by sftpwrapper.
12. The LogM Library updates the `reportProgress` attribute of the corresponding Log instance with the final result of the operation.

The progress report can be seen in the NBI.

13. The LogM Library calls the LogMgmtController API callback `export_completed_callback(LogStreamName)`.
14. The Log Controller removes the temporary tar file used for export.
15. The result is returned back to the LogM Library.

Note: Step 10 is repeated periodically some times until the export operation is complete or failed. The last `reportProgress` indicates the outcome.

This use case assumes valid `uri`, `password`, and `transferType` to perform export. If invalid, Step 9 immediately gives a failure result and the same steps occur to notify the NBI user by the `reportProgress` attribute.

If a `cancel()` operation is started by the NBI user on a Log instance during an ongoing `export()` operation, the same steps occur. The LogM Library creates a child process for sftpwrapper and the `cancel()` action causes this child process to be killed.



6 Process View

6.1 Core MW LogM Users and Groups

The following users and groups are reserved for Core MW LogM:

- `cmw-logm` (Group ID: 361)
- `cmw-logm` (User ID: 330)

The `cmw-logm` user is also part of the following supplementary groups (if available) to interact with other CBA components:

- `cmw-imm-users`
- `cmw-log-data`
- `swm`
- `system-nbi-data`

6.2 Core MW LogM Processes

If LDE supports processes running with least privileges, the following Core MW LogM process is executed as non-root user:

- `cmwlogm`

This process is the Core MW LogM framework. This includes the Core MW Log Controller, which uses the shared LogM Library.

- The LogM Library must be added to the `cmw-imm-users` group to use the IMM API.
- The LogM Library must be added to the `swm` group to use `sftpwrapper` to perform export.
- The Core MW Log Controller must be added to the `cmw-log-data` group to view OpenSAF log streams
- The Core MW Log Controller must be added to the `system-nbi-data` group to be able to export log streams registered by COM.





7 ECIM Security Roles and Rules

The CBA component COM delivers role instances for: `SystemAdministrator` and `SystemSecurityAdministrator`, and `SystemReadOnly`.

For a description of COM Security Management, refer to COM Management Guide.

Core MW LogM delivers rule instances for the Management Object Model (MOM) fragments it implements (LogM) for the roles `SystemAdministrator` and `SystemSecurityAdministrator`, and `SystemReadOnly`. These rule instances are delivered using Core MW MDF.

7.1 System Administrator Role

A system administrator is responsible for the administration of all non-security-related attributes and capabilities of a managed element, for example, managed element features, capabilities, configuration parameters, and monitoring of the managed element.

The rules delivered, to provide the system administrator with proper access to model fragments implemented, are the following:

- Full access to the LogM MO and all MOs below, including all attributes and actions.
- Full access to the Log=saLogSystem MO and all MOs below, including all attributes and actions.

7.2 System Security Administrator Role

A system security administrator is responsible for the administration of the attributes and capabilities of a managed element related to security of the managed element, regardless of which applications that execute on the managed element, for example, managed element administrative, user accounts, and authorizations.

The rule delivered, to provide the system security administrator with proper access to model fragments implemented, is the following:

- Read access to the LogM MO, including all attributes.

7.3 System Read Only Role

A system read only is responsible for the monitoring of system configuration parameters.



The rules delivered, to provide the system administrator with proper access to model fragments implemented, are the following:

- Read access to the LogM MO, including all attributes and actions.
- Read access to the Log=saLogSystem MO and all MOs below, including all attributes and actions.



8 Deployment View

This section describes the Core MW LogM configuration to physical nodes. Core MW LogM is deployed as a CBA component and requires Core MW.

8.1 Redundancy

Core MW LogM contains the following components:

- The Core MW Log Controller component, which runs as `cmwlogm` process, is deployed with a 2-N redundancy model and runs on System Controllers only.
- The LogM Library component does not have a redundancy model and is explicitly allocated on all System Controller nodes. It may be implicitly allocated on other nodes if services specify a dependency towards the Library.

8.2 Failover and Switchover Operations

8.2.1 Failover

Failover is a recovery operation performed by the AMF when an active Core MW Log Controller component fails.

8.2.2 Switchover

Switchover is an operation performed by the AMF to make the standby Core MW Log Controller component active.

8.3 Linux System Solutions

Core MW LogM supports the following Linux system solutions:

- Linux disk-based root file system
- Linux RAM-based root file system

8.4 Alarms

Core MW LogM provides no alarms.





9 Implementation View

9.1 Overview

An overview of the top level of the Core MW LogM source code structure is listed in Table 3. The source code is managed in a Git version system.

Table 3 LogM Source Code Structure

Parent Folder	Subfolder	Description
src/cmwlogm	common/	Common source code used by both library and controller
	doc/	Doxygen documentation related to LogMgmt APIs
	lib_logm/	LogM Library source code
	log_mgmt_api/	Log Management API headers
	samples/	LogMgmt API examples
	models_cntlr/instances/src	Papyrus source file of saLogSystem Log Controller Log instance
	models_cntlr/instances/output_models	saLogSystem model instance file packaged by Log Controller
	models/cmw_logm/	Model source files to generate LogM classes
	models/instances	Model source files to generates LogM instances
	sudoers/	
	trace_files/	



Table 3 LogM Source Code Structure

Parent Folder	Subfolder	Description
sdp/cmwlogm	deploy/	Packaging of LogM deployment archive including CSM model
	log_controller_bundle/	Packaging of Cmw Log Controller bundle SDP including AMF scriptPackaging of Cmw Log Controller bundle SDP including AMF script
	logm_lib_bundle/	Packaging of LogM Library bundle SDP including CmwLogM models
	runtime/	Packaging of LogM runtime archive
	sdk/	Packaging of LogM SDK and OAM models archive
	debug/	

9.2 Logging and Trace

Core MW LogM logs system-important events to syslog.

CoreMW LogM is instrumented for logging via Trace CC.



10 Data View

Core MW LogM stores data that needs to be persisted within the CmwLogM model, and the instances are stored by IMM.





11 Size and Performance

For information on performance and characteristics test cases, refer to Performance Tests of Core MW.





12 Quality

The quality of Core MW LogM has been tested according to the Core MW LogM System Test Specification.

For more information, refer to Core MW LogM System Test Specification.





13 Third-Party Components

— libcrypto

Core MW LogM makes use of libcrypto provided by LDE.