Standard Diagnostic Logging

Version 5.0

23 September 2021

This is a Non-binding Permanent Reference Document of the GSMA

Security Classification: Non-confidential

Access to and distribution of this document is restricted to the persons permitted by the security classification. This document is subject to copyright protection. This document is to be used only for the purposes for which it has been supplied and information contained in it must not be disclosed or in any other way made available, in whole or in part, to persons other than those permitted under the security classification without the prior written approval of the Association.

Copyright Notice

Disclaimer

The GSM Association ("Association") makes no representation, warranty or undertaking (express or implied) with respect to and does not accept any responsibility for, and hereby disclaims liability for the accuracy or completeness or timeliness of the information contained in this document. The information contained in this document may be subject to change without prior notice.

Compliance Notice

The information contain herein is in full compliance with the GSM Association's antitrust compliance policy.

This Permanent Reference Document is classified by GSMA as an Industry Specification, as such it has been developed and is maintained by GSMA in accordance with the provisions set out in GSMA AA.35 - Procedures for Industry Specifications.

Table of Contents

1 Introduction

1.1 Overview

The purpose of this document is to provide a standardized method to log modem data and messaging on a device, eliminating the need for tethered logging. The primary user of the logging tool is expected to be mobile network operators.

Possible use cases are listed below:

- Report the geo-location of the device and key RF parameters (RSRP, RSSI, SINR etc.) to determine network coverage
- Present geo-located events on maps to allow better call drop analysis
- Capture handover statistics to debug handover issues
- Report VoLTE (Voice over LTE) call statistics (e.g. Delay, Jitter and Packet Loss) to aid in VoLTE analysis
- Real time reporting on the device
    - Data throughput
    - Txpower
    - Cell selection
    - RF parameters

The operator will be able to log with any device/chipset compliant to the interface outlined in this document by downloading a compliant application. The logs will be saved on the device and uploaded to an operator server. This document provides the APIs (Application Programming Interfaces) and MIB (Management Information Base) to capture the modem and other components' log data; and the security protocol to authenticate the device before logging can be initiated. The API source code is available on the GSMA GitHub site, the manufacturers and application vendors are encouraged to download this source code to implement the standard logging solution.

The API defined in this document to retrieve the modem and other components' log data are part of the MDMI (Modem Diagnostic Monitoring Interface). Using the MDMI, the diagnostic application will be able to retrieve information from the components being logged, such as a KPI or a protocol message. MDMI is modelled on and is defined using a standardized format, SNMP (Simple Network Management Protocol). As per SNMP convention, all information retrieved from the components being logged, is passed as objects and are defined in the MIB. There are some advantages of using this framework to define the interface to the modem. However, MDMI diverges from the SNMP specification in several ways. In particular, MDMI cannot be implemented by the use of SNMP tools without additional effort.

This document is organized as follows:

Section 2 introduces the MDMI (Modem Diagnostic Monitoring Interface) and presents an overview of how it utilizes SNMP (Simple Network Management Protocol). This overview includes illustrations of support for multiple diagnostic feeds.

Note: The previous version of this specification illustrated an architecture for logging the modem component alone.

Section 3 presents architectures on the device to implement the standard diagnostic interface

Section 4 describes how SNMP has been adapted for use in MDMI, including an overview of the types of information that MDMI makes available, the relationships between the different application components, and method that is used for exchanging messages.

Section 5 outlines MDMI, including the functions that must be implemented, and provides an overview of the MIB (Management Information Base), which defines all the messages, KPIs (Key Performance Indicators) and commands that must be made available.

Section 6 describes the Security Architecture Design for MDMI, including both the authentication of the device required before logging can take place, and the security of the log data. Authentication methods for both an on board tool and remote log session control are disucssed.

Annex A References the UICC/eUICC white list.

Annex B References the source code location.

Annex C Document History.

1.2 Scope

The initial scope of the GSMA standard diagnostic logging interface (version 1.0) was restricted to engineering builds for LTE and Wi-Fi only. Scope is now expanded to include additional technologies such as Wi-Fi calling, eMBMS, IMS, UICC/eUICC. Further expansion of the scope requires further study.

1.3 Definitions

Term	Description
ASN.1 BER	The encoding used to pack a Log Record, as in SNMP
DM App	Diagnostic Monitoring Application - any app that uses MDMI
Commercial Build	Software that is available to an end user or customer.
Engineering Build	Software provided by an OEM to a network operator for the sole purpose of testing by the network operator and its representatives, and not for release to customers.
Event	A Log Record pushed by MDMI to a DM App
KPI	Key Performance Indicator - KPI Log Records report KPIs
Log Record	A single piece of diagnostic information from MDMI, either pulled from MDMI by a DM App, or pushed by MDMI to the DM App
MDMI	Modem Diagnostic Monitoring Interface - interface defined by this document
MDMI Session	An identifier MDMI uses to identify a particular DM App - this will be assigned during MDMI initialization
MDMI Value	The contents of a Log Record - either KPIs or a protocol message - encoded

Term	Description
	using ASN.1 BER into a buffer of bytes
MIB	Management Information Base – set of diagnostic objects that can be managed by SNMP
OID	Object ID – the unique identifier for an SNMP object
Protocol Message	A Log Record containing an OTA message of a particular protocol
SNMP	Simple Network Management Protocol
SNMP Agent	The provider of diagnostic information - in MDMI, the OEM writes a library conforming to this requirement that acts as an SNMP agent
SNMP Manager	The user of SNMP Agent - in MDMI, this would be a component of a DM app
SNMP Object	A unit of diagnostic information defined in the MIB
SNMP Trap	SNMP terminology for a message pushed from agent to manager - in MDMI, we use the term "Event" synonymously

1.4 Abbreviations

Term	Description
API	Application Programming Interface
ASN	Abstract Syntax Notation
BDN	Barred Dialling Numbers
BER	Basic Encoding Rules
DM	Diagnostic Monitoring
eMBMS	Evolved Multimedia Broadcast Multicast Services
eUICC	A removable or non-removable UICC which enables the remote and/or local management of Profiles in a secure way
FDN	Fixed Dialling Numbers
GUTI	Globally Unique Temporary Identifier
HMAC	Hash Message Authentication Code
IPC	Inter-Process Communication
IMSI	International Mobile Subscriber Identity
IMEI	International Mobile Equipment Identity
JSON	Java Script Object Notation
KPI	Key Performance Indicator
NAS	Non-Access Stratum
MCC	Mobile Country Code
MDMI	Modem Diagnostic Monitoring Interface
MIB	Management Information Base
MNC	Mobile Network Code
MSIN	Mobile Subscriber Identification Number
OEM	Original Equipment Manufacturer

Term	Description
OID	Object ID
P-TMSI	Packet – Temporary Mobile Subscriber Identity
PDU	Protocol Data Unit
PIN	Personal Identification Number
PUK	Personal Identification Number Unlock Key
PUSCH	Physical Uplink Shared Channel
RSRP	Reference Signal Received Power
RSRQ	Reference Signal Received Quality
RSSI	Received Signal Strength Indicator
RTCP	RTP Control Protocol
RTP	Real Time Transfer Protocol
RRC	Radio Resource Control
SINR	Signal to Noise Ratio
SIP	Session Initiation Protocol
SNMP	Simple Network Management Protocol
SW	Software
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TMSI	Temporary Mobile Subscriber Identity
UDP	User Datagram Protocol
UICC	Universal Integrated Circuit Card
UTF	Unicode Transformation Format

1.5 References

Several standards were used to develop MDMI. They are listed here as a reference.

References	
SNMP	
SNMP standards were used to define the MIB and messaging format.	
Structure and Identification of Management Information	http://www.ietf.org/rfc/rfc1155.txt
SNMP	http://www.ietf.org/rfc/rfc1157.txt
MIB	http://www.ietf.org/rfc/rfc1212.txt
MIB-2	http://www.ietf.org/rfc/rfc1213.txt
SNMP Traps	http://www.ietf.org/rfc/rfc1215.txt
3GPP	

References	
All logged messages should be reported in their original format without modification, as described by the 3GPP standard.	
RRC	36.331 - Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Resource Control (RRC); Protocol specification
NAS	24.301 - Non-Access-Stratum (NAS) protocol for Evolved Packet System (EPS)
MAC	36.321 - Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification
PDCP	36.323 - Evolved Universal Terrestrial Radio Access (E-UTRA); Packet Data Convergence Protocol (PDCP) specification
RLC	36.322 - Evolved Universal Terrestrial Radio Access (E-UTRA); Radio Link Control (RLC) protocol specification
TS 31.121	UICC-terminal interface; Universal Subscriber Identity Module (USIM) application test specification
TS 31.124	Mobile Equipment (ME) conformance test specification; Universal Subscriber Identity Module Application Toolkit (USAT) conformance test specification
TS 34.108	Common test environments for User Equipment (UE); Conformance testing
TS 51.010	Digital cellular telecommunications system; Mobile Station (MS) conformance specification;
IP	
IP Packets (and all protocol messages contained therein) logged should also be in their original format without modification, as described by IETF.	
IPv4	http://www.ietf.org/rfc/rfc791.txt
IPv6	http://www.ietf.org/rfc/rfc2460.txt
ETSI	
TS 102 221	http://www.etsi.org/deliver/etsi_ts/102200_102299/102221/
TS 102 223	http://www.etsi.org/deliver/etsi_ts/102200_102299/102223/
TS 102 613	http://www.etsi.org/deliver/etsi_ts/102600_102699/102613/
GSMA	
TS.26 NFC Handset Requirements	https://www.gsma.com/newsroom/gsmadocuments/
PDATA.12	http://www.gsma.com/identity/wp-content/uploads/2017/01/PDATA.12-SIM-Toolkit-Device-Requirements-to-improve-Mobile-Connect-Customer-Experience-v1.0.pdf

References	
SGP.23	https://www.gsma.com/newsroom/all-documents/sgp-23-v1-0-rsp-test-specification/
ISO	
ISO/IEC 7816-3	Identification cards -- Integrated circuit cards -- Part 3: Cards with contacts -- Electrical interface and transmission protocols
ISO/IEC 7816-4	Identification cards -- Integrated circuit cards -- Part 4: Organization, security and commands for interchange

2 Standard Diagnostic Interface Overview

The standard logging interface is referred to in this document as the Modem Diagnostic Monitoring Interface (MDMI), which is an application programming interface (API) and a messaging interface between Diagnostic Monitoring (DM) applications running on a mobile device and the device component being logged. The modem is the main component that is logged. MDMI enables DM applications to monitor and control the activities of the component being logged. The messaging interface is based on a modified version of the Simple Network Management Protocol (SNMP).

Table 1 outlines an example use case of primary functions of this interface, and how SNMP is used to achieve this function.

Function	Example	Mechanism
Interrogate	What is the device's current RSRP?	SNMP GetRequest
Configure	Set Airplane Mode On	SNMP SetRequest
Command	Make a telephone call	SNMP SetRequest
Log Subscribe	Subscribe to all RRC Messages	Specify Mask: SNMP SetRequest Receive Information: Event (SNMP Trap)

Table 1 Primary Functionality

The information that can be retrieved through MDMI can be described as SNMP objects, or Log Records. Types of Log Records are listed in Table 2

Type	Examples	Pushed/Pulled	Format Definition
KPI Log Record	• RSRP • PUSCH Tx Power • Path Loss	Push or Pull	Defined in MIB
Protocol Message Log Record	• RRC Messages • NAS Messages • IP Packets	Push	• Header defined in MIB • Payload defined by relevant standard
Command Result Log	• Success of Phone Call	Push	Defined in MIB

Record	• Location determined by Fix		
Configuration Log Record	• Device Name • MDMI Version	Pull	Defined in MIB

Table 2 Log Record Types

As illustrated in Figure 1, the device OEMs are expected to implement an SNMP agent that provides the MDMI programming interface specified in this document. Access to the interface may be restricted to DM applications which are approved by the network operator receiving the engineering build (see Section 6). The operating system may prohibit access to the interface by applications which are not approved by the network operator. DM applications developed by third parties utilize the MDMI programming interface to monitor and control the device component being logged. As an example, when invoked by a DM application using MDMI, the SNMP Agent in the application processor (A-processor) could interact with the modem in the B-processor via inter-process communication (IPC) to perform the monitoring and control functions requested by the DM application. MDMI is generically defined with the goal that it is implementable by an OEM, regardless of the device operating system or the device chipset. As such, a DM application using MDMI should run on any such device without modification.

2.1 MDMI Architecture Supporting Multiple Feeds

Figure 1 illustrates possible modem chipset architecture with multiple data sources and an MDMI Agent for each source. All pairs of source and agent support the same MDMI interface as currently specified by either through native MDMI.h file or through java IMdmiInterface.aidl file.

DM Application discovers the Agent library by consulting the MIB Discovery Database.

DM applications developed by third parties will develop the MDMI Manager to link to each of the multiple MDMI Agents and will utilize the MDMI interfaces to monitor and control the each of the data sources.

When the DM application invokes a data source’s MDMI agent, using MDMI, the MDMI agent in the application processor (A-Processor) will then interact with its corresponding data source (e.g. Modem, Wi-Fi, or eMBMS middleware) to perform the monitoring and controlling functions typically requested by an MDMI DM application.

The data requested by the DM application and collected by the MDMI agents will be returned to the MDMI manager through the corresponding MDMI Interface. How to handle the incoming traffic from multiple feeds is out of scope of this specification.

The MDMI agents are required to implement only those OIDs from the MDMI MIB tree that are relevant to the data source they handle. In Figure 2, this means that:

- The MDMI LTE Modem Agent will log OIDs corresponding to the LTE branch of the MIB tree based on the MIB Discovery Database

- The MDMI Wi-Fi Agent will log OIDs corresponding to the Wi-Fi branch of the MIB tree based on the MIB Discovery Database.
- The MDMI eMBMS Agent will log OIDs corresponding to the eMBMS branch of the MIB tree based on the MIB Discovery Database.
- The MDMI IMS Agent will log OIDs corresponding to the IMS branch of the MIB tree based on MIB Discovery Database.
- The MDMI Modem Agent can log the OID's corresponding to both the LTE and IMS branches of MIB tree based on MIB Discovery Database.
- The same apply to other branches of the MIB tree

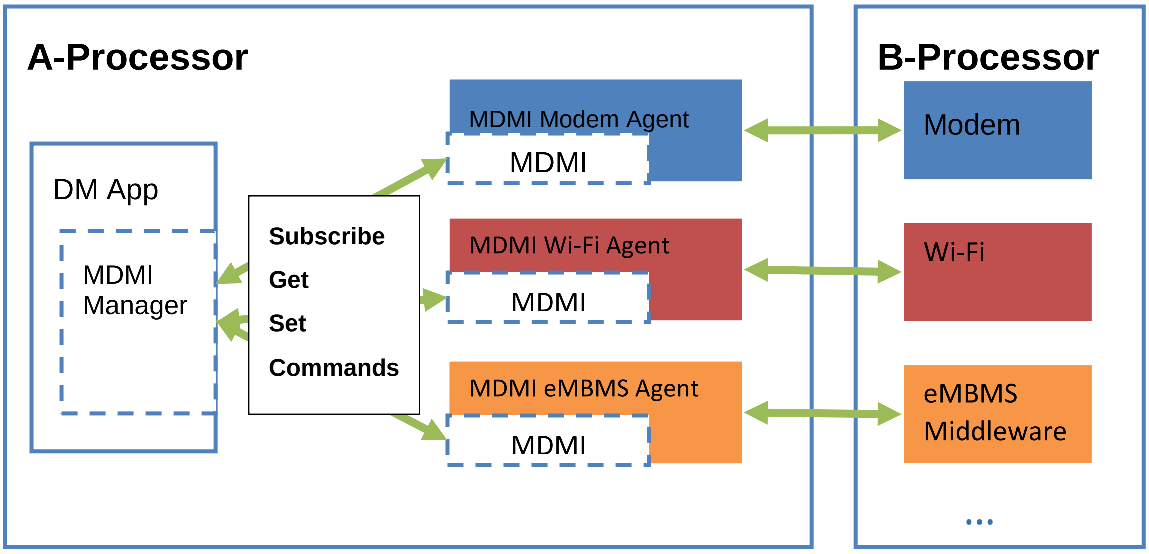


Figure 1 Example of Chipset Architecture with Multiple Sources and Agents

Figure 2 illustrates the generic device architecture, with support for a variety of sources: multiple sources within one chip, sources from multiple chips, and sources within the A-processor. In all cases, each source interfaces with its own MDMI Agent.

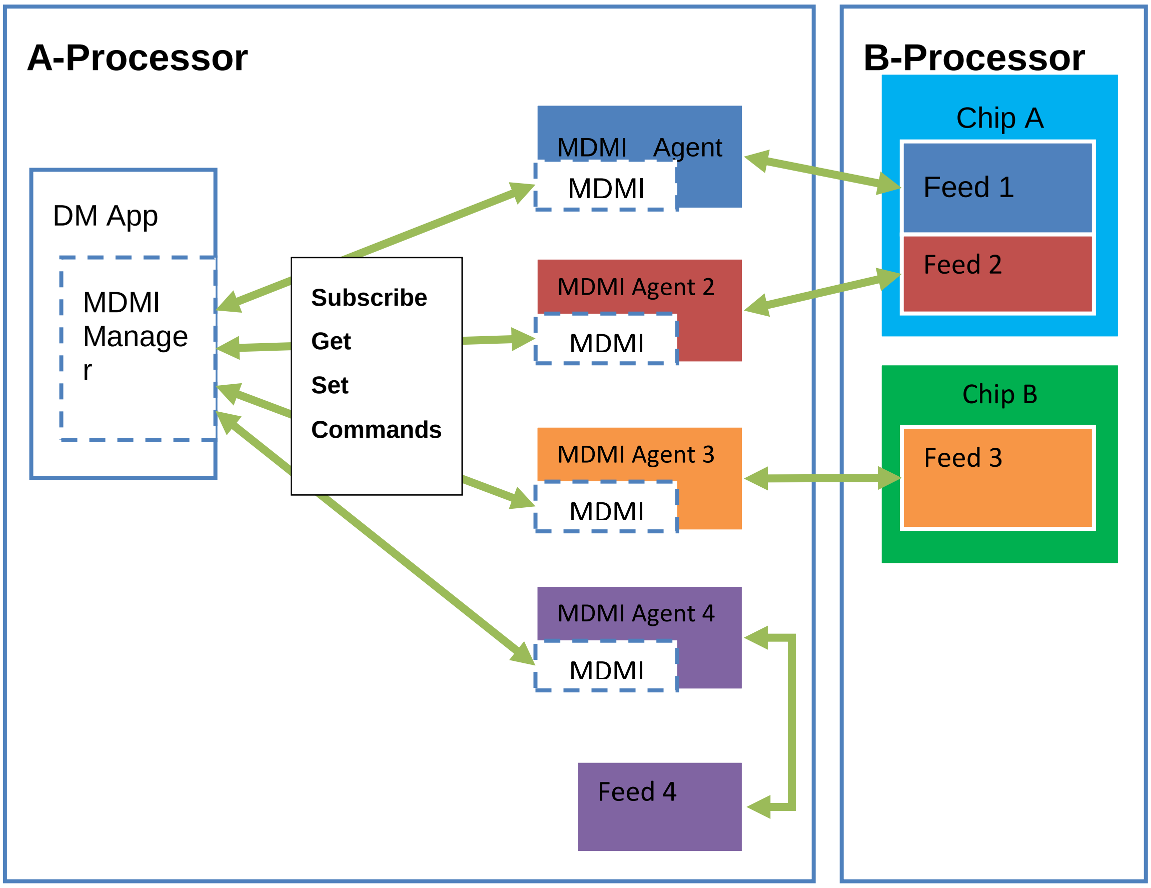


Figure 2 Generic Device Architecture

The requirements for MDMI multiple feeds are summarized as follows:

1. Individual MDMI Agents are needed in the A-Processor for logging each data source, whether from the A-Processor or from the B-Processor
2. Multiple data sources from a single chip in the B-Processor require separate MDMI Agents for each feed
3. Each MDMI Agent will be implemented either in java or native. DM App discovers the Agent library name by reading the MIB Discovery Database(com.gsma.mdmi.db)
4. When Agent is implemented in native, the agent provides a unique .so library. The exposed .so library implements the MDMI interface, as specified in the header file (MDMI.h)
5. When Agent is implemented in java, the agent provides a java jar file. The jar file exposes an AIDL interface (IMdmiIInterface.aidl) The Agent implements the AIDL interface in a service. DM App binds to the service and issues the AIDL calls to the MDMI interface

2.1.1 UICC/eUICC specific logging

UICC/eUICC logging has been added as one of the MDMI feeds. This enables logging between the UICC/eUICC and the device baseband. The interface between UICC and the device is defined by the contacts C6 and C7 of the UICC. Two different protocols are provided with these contacts and these are the general UICC communication protocol (ISO/IEC 7816-3, 7816-4) between UICC and the device baseband and the specific SWP (Single Wire Protocol) between UICC and CLF (ContactLess Frontend, NFC) (ETSI TS 102 613).

For the ISO interface, APDU commands and responses between the UICC and device baseband are logged. SWP protocol logging is not added to MDMI at this time and will be re-visited later.

An eUICC contains overall the same functionalities as the UICC with additional features introduced. The basic logging will cover traditional UICCs as well as eUICCs. The structure and format of APDU commands and responses being logged are defined in ETSI TS 102 221 and TS 102 223.

2.1.2 Data White-listing

APDU logging can result in critical information being captured in the logs or being exposed outside the baseband. There are several types of data which shall be protected and not included in the logging –

1. Data belonging to the user is sensitive and should not be logged. Examples include phonebook (includes user PIN/PUK), SMS and call logs (FDN, BDN, time of call & duration of incoming/outgoing calls etc.). Exposing this data in logs that are uploaded externally may cause privacy concerns. In general, many of the writable EFs in the UICC fall in this category, with a few exceptions.
2. Some data also needs to be restricted to the modem. Main concern relates to the result of authentication algorithm (Ck and Ik). Some other examples of parameters that shall be masked are TMSI, P-TMSI, P-TMSI signature, GSM Ciphering key Kc, GPRS Ciphering key KcGPRS, GUTI and parameters related to WLAN authentication and identity.
3. Care must be taken to guard the integrity of STK exchanges as specified by GSMA doc PDATA.12.
4. GSMA TS.26 has the requirement that the device shall not log any APDU or AID exchanged in a communication with an applet located on the SE.
5. Some of the Toolkit – Proactive Commands contain sensitive data that shall not be logged. This includes all commands/responses related to user key input, menus, multimedia etc.

Integrity of the sensitive data shall be preserved by masking the sensitive data in the payload. Annex A includes a list of data which shall be excluded from the logging.

For engineering builds using Test UICCs or eUICC Test Profiles, data shall not be masked, since all debugging and logging information may be needed.

The following IMSI values SHALL enable full logging of UICC and eUICC data: (IMSI logically values, EF 6F 07, 3GPP TS 31.102 section 4.2.2)

MCC	MNC (2 or 3 digits)	MSIN	Reference specification

MCC	MNC (2 or 3 digits)	MSIN	Reference specification
001	Any value	Any value	3GPP TS 51.010-1, A4.3.3 3GPP TS 31-124, 27.22.2a
246	81	3579	3GPP TS 31.121, 5.1.2.4.1
246	081	3579	3GPP TS 31.121, 4.1.1.1
246	81	1111111111	3GPP TS 51.010-1, 27.4.4.1
246	813	111111111	3GPP TS 51.010-1, 27.4.4.1
246	81	3579	3GPP TS 51.010-1, 27.10a.4.1 3GPP 31.124 27.22.4.7.2.4/5
246	81	357X	3GPP TS 51.010-1, 27.10a.4.1
442	01	Any value	3GPP TS 34.108, 8.3.2.2 plus restriction in sect 8.3.2.2
299	811	1234 56789	GSMA SGP.23, A.1
299	821	1234 56779	
299	821	1234 56769	
299	843	4567 89012	
299	811	1234 56789	
299	811	1234 56779	
299	883	4567 89012	
299	893	4567 89012	

X: Means any value in the range 0 to 9.

2.1.3 Logging during power-up

The modem starts initialization of the UICC/eUICC very early, likely before any pipe is established between the Application Processor for logging APDUs with SDL. This may lead to loss of power-up logs. Additionally it may not be possible for an application or user to restart the UICC/eUICC.

The SDL implementations shall incorporate mechanisms to prevent the power-up logs from being lost since these are one of the most critical components of UICC/eUICC logging. Caching of the power up logs at the modem is one possible solution, but it is implementation dependent.

2.2 MIB Discovery Database

DM App consults the MIB Discovery database to discover the Agent name for a feed based on the Category OID. Category OID specifies the parent OID. DM App consults the corresponding Agent library name for all the child OID’s in the Category OID.

This data base has two columns <Category Oid, Agent Name>

Examples of tuples in the Database can be as below

<”1.1”, “lib_mdmi_debug.so”>
<”1.2”, “lib_mdmi_lte.so”>
<”1.3”, “lib_mdmi_wcdma.so”>
<”1.4”, “lib_mdmi_embms.so”>
<”1.6”, “com.gsma.mdmi.ImsService.jar”>
<”1.7”, “com.gsma.mdmi.WifiService.jar”>
<”1.8”, “lib_mdmi_gsm.so”>
<”1.9”, “lib_mdmi_umts.so”>
<”1.10”, “lib_mdmi_ltemiddleware.so”>
<”1.11”, “lib_mdmi_hsupa.so”><”1.12”, “lib_mdmi_hsdpa.so”>

3 Cross Platform Compatibility

MDMI is generically defined with the goal that it is implementable by one or more chip vendors, regardless of the device operating system or the device chipset. Consequently, a DM application using MDMI should run on any such device without modification.

This library has been designed to make integrating a DM app onto devices as seamless as possible. To the extent possible, the usage should be identical across devices, chipsets, OEMs and even operating systems. However, due to underlying differences between current mobile operating systems, some differences will be inevitable.

Figure 3 shows an example of an Android implementation of the architecture in Figure 1. Figure 4 shows an example of implementing the same architecture on an alternate platform. In both operating systems, the OEM will provide a library making all the functionality of MDMI available to a DM app. On the Android, the .so libraries will be pre-installed on the system. A DM app, once installed, will dynamically link to the libraries to use the functionality of MDMI. On alternate platforms, pre-installing the library may not be possible, due to platform restrictions. The libraries will be prepared by the chipset vendors and provided to the developer of the DM App as dll files. The dll files will be compiled directly into the DM app. The DM App will have to be recompiled for each version of the libraries.

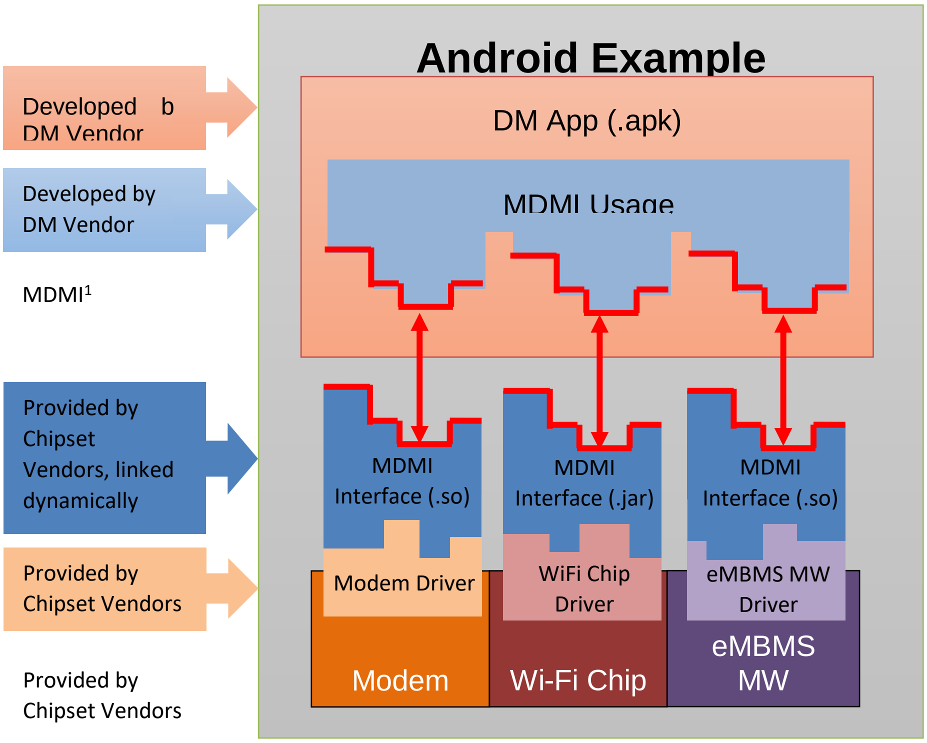


Figure 3 Example of Android Implementation

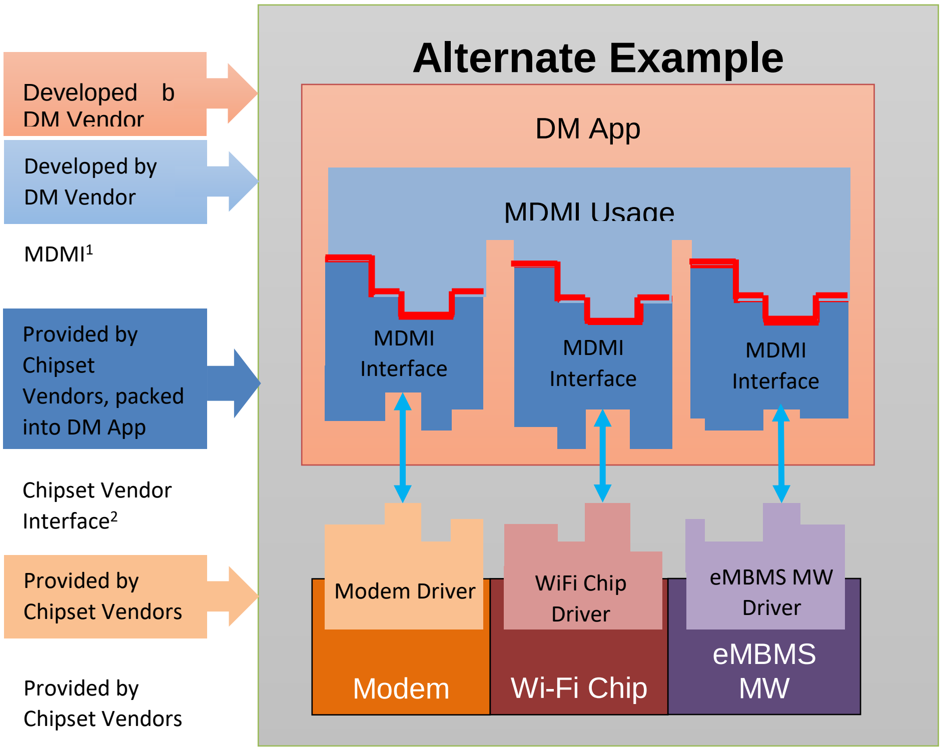


Figure 4 Example of Alternate Implementation

4 Use of SNMP

The MDMI interface is based on SNMP, in spite of their differences (notably, that the interface is not over UDP/IP). The reason for basing the interface on SNMP is to make use of a standardized monitoring and control interface structure. As described in Section 3, the DM application includes the functionality of a SNMP Manager, and the MDMI module, which provides a programming interface to extract modem information, acts as a SNMP Agent.

The desired implementation can be achieved with SNMPv1 and, only a subset of SNMPv1 is used.

The MIB defines all log objects that are available through MDMI. These are organized hierarchically, and each object has an OID as identifier. Some are available to be read, and others can be both read and written. Some are available to be pushed to the DM App as events once the DM App has subscribed to them. The MIB also defines the exact syntax of each field in each object. The scope of these objects covers a set of KPIs and protocol messages, as well as some basic configuration items and commands. The design is extensible so that more KPIs, protocols and commands can be added in future releases.

4.1 Method of SNMP Message Exchange

SNMP operates over UDP. MDMI replaces this with function calls in which a buffer is passed from the DM App to the SNMP Agent and vice versa. This buffer will contain the SNMP requests/responses encoded in the same way SNMP is encoded (ASN.1 BER encoding scheme, see references in section 1 for more details). Both open source and commercial libraries exist to encode and decode ASN.1, and are widely used in many other telecommunication protocols.

An application can either pull logs from MDMI by using a "Get" message, or the application can request that logs to be pushed to it by specifying which events should be sent from the MDMI to the DM App. When events are to be pushed, the DM App must specify the function MDMI will use as a pointer using `MdmiSetEventCallback`, and specifying the events that are requested using `MdmiSubscribe`.

5 MDMI

The basics of the API are described below. Devices implementing MDMI must conform to these definitions precisely.

5.1 MDMI Native Interface

Each required function is listed in the tables below with:

- Function Name - the name of the function
- Signature - the exact function signature that must be implemented
- Arguments - name and explanation of the arguments passed to the function
- Return Value - value returned by the function

5.1.1 MdmiCreateSession

This function creates a MDMI session that is used in subsequent MDMI calls to identify the caller.

Function Name	**MdmiCreateSession**

Signature	`MdmiError` **`MdmiCreateSession`**
	`(const wchar_t* address,`
	`MdmiSession* session);`

Arguments	`address`: address of the MDMI device to open. May be ignored if the system has only one device `session`: session object that will be set upon success. This will be used by the caller in subsequent calls to MDMI.
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.2 MdmiCloseSession

This function closes the MDMI session.

Function Name	**MdmiCloseSession**

Signature	`MdmiError` **`MdmiCloseSession`**
	`(MdmiSession* session);`

Arguments	`session`: session object that will be closed. If close is succesful, the session object is set to 0, indicating invalid session
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.3 MdmiGet

This function gets the value of a specific object, as specified by that object's OID.

Function Name	**`MdmiGet`**

Signature	`MdmiError` **`MdmiGet`**
	`(MdmiSession session,`
	`const MdmiObjectName* name,`
	`MdmiValue* value);`

Arguments	`session:` identifies the session `name:` OID of the value `value:` value to be read. If the read is succesful, the actual value is read into this pointer. Upon return the ownership of this pointer is transferred to caller and must be freed when no longer needed
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.4 MdmiSet

This function sets the value of a specific object, as specified by that object's OID.

Function Name	**`MdmiSet`**

Signature	`MdmiError` **`MdmiSet`**
	`(MdmiSession session,`
	`const MdmiObjectName* name,`
	`const MdmiValue* value);`

Arguments	`session:` identifies the session `name:` OID of the value `value:` value to be set
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.5 MdmiInvoke

This function invokes a command through MDMI. Commands are defined in the MIB and identified by an OID.

Function Name	**MdmiInvoke**

Signature	`MdmiError` **`MdmiInvoke`**
	`(MdmiSession session,`
	`const MdmiObjectName* name,`
	`const MdmiValue* value);`

Arguments	`session:` identifies the session `name:` OID of the command to invoke `value:` optional value of the command (can be null)
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.6 MdmiSetEventCallback

This function sets the call back function that will be used for pushed events.

Function Name	**MdmiSetEventCallback**

Signature	`MdmiError` **`MdmiSetEventCallback`**
	`(MdmiSession session,`
	`MdmiEventCallback callback,`
	`void* state);`

Arguments	`session:` identifies the session `callback:` The callback function pointer. This value will replace previous value. Setting this value to NULL will stop event callbacks. See MDMI.h for definition of the callback. `state:` Optional state that will be passed when callback function is called
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.7 MdmiSubscribe

This function specifies an object, which should be reported via trap message whenever it is updated.

Function Name	`MdmiSubscribe`

Signature	`MdmiError` **`MdmiSubscribe`**
	`(MdmiSession session,`
	`const MdmiObjectName* name);`

Arguments	`session:` identifies the session `eventName:` identifies the event to be registered. Multiple registrations will still result in only one event being generated
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.8 MdmiUnsubscribe

This function removes an object from the list of objects that should be reported via trap message.

Function Name	`MdmiUnsubscribe`

Signature	`MdmiError` **`MdmiUnsubscribe`**
	`(MdmiSession session,`
	`const MdmiObjectName* name);`

Arguments	`session:` identifies the session `eventName:` identifies the event to be deregistered
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.1.9 MdmiGetSessionStats

This function gets the statistics of the session.

Function Name	`MdmiGetSessionStats`

Signature	`void` **`MdmiGetSessionStats`**
	`(MdmiSession session, MdmiSessionStats* stats);`

Arguments	`session:` identifies the session `stats:` the statistics to be returned
Return Value	`MDMI_NO_ERROR on success otherwise an error`

5.2 MDMI Java Interface

Each required function is listed in the tables below with:

- Function Name - the name of the function
- Signature - the exact function signature that must be implemented
- Arguments - name and explanation of the arguments passed to the function
- Return Value - value returned by the function

5.2.1 MdmiCreateSession

This function creates a MDMI session that is used in subsequent MDMI calls to identify the caller.

Function Name	`MdmiCreateSession`
Signature	`int MdmiCreateSession(in String address, out MdmiSession session);`
Arguments	`address`: address of the MDMI device to open. May be ignored if the system has only one device `session`: session object that will be set upon success. This will be used by the caller in subsequent calls to MDMI.
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.2.2 MdmiCloseSession

This function closes the MDMI session.

Function Name	`MdmiCloseSession`
Signature	`int MdmiCloseSession(in MdmiSession session);`
Arguments	`session`: session object that will be closed. If close is succesful, the session object is set to 0, indicating invalid session
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.2.3 MdmiGet

This function gets the value of a specific object, as specified by that object's OID.

Function Name	MdmiGet
Signature	`int MdmiGet(in MdmiSession session, in MdmiObjectName name, out MdmiValue value);`
Arguments	`session:` identifies the session `name:` OID of the value `value:` value to be read. If the read is succesful, the actual value is read into this pointer. Upon return the ownership of this pointer is transferred to caller and must be freed when no longer needed
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.2.4 MdmiSet

This function sets the value of a specific object, as specified by that object's OID.

Function Name	MdmiSet
Signature	`int MdmiSet(in MdmiSession session, in MdmiObjectName name, in MdmiValue value);`
Arguments	`session:` identifies the session `name:` OID of the value `value:` value to be set
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.2.5 MdmiInvoke

This function invokes a command through MDMI. Commands are defined in the MIB and identified by an OID.

Function Name	MdmiInvoke
Signature	int MdmiInvoke(in MdmiSession session,in MdmiObjectName name,in MdmiValue value);
Arguments	session: identifies the session name: OID of the command to invoke value: optional value of the command (can be null)
Return Value	MDMI_NO_ERROR on success otherwise an error

5.2.6 MdmiSetEventCallback

This function sets the call back function that will be used for pushed events.

Function Name	MdmiSetEventCallback
Signature	int MdmiSetEventCallback(in MdmiSession session, in IMdmiEventCallback callback, in CallbackState state);
Arguments	session: identifies the session callback: The callback function pointer. This value will replace previous value. Setting this value to NULL will stop event callbacks. See MDMI.h for definition of the callback. state: Optional state that will be passed when callback function is called
Return Value	MDMI_NO_ERROR on success otherwise an error

5.2.7 MdmiSubscribe

This function specifies an object, which should be reported via trap message whenever it is updated.

Function Name	**`MdmiSubscribe`**
Signature	`int MdmiSubscribe(in MdmiSession session, in MdmiObjectName eventName);`
Arguments	`session:` identifies the session `eventName:` identifies the event to be registered. Multiple registrations will still result in only one event being generated
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.2.8 MdmiUnsubscribe

This function removes an object from the list of objects that should be reported via trap message.

Function Name	**`MdmiUnsubscribe`**
Signature	`int MdmiUnsubscribe(in MdmiSession session, in MdmiObjectName eventName);`
Arguments	`session:` identifies the session `eventName:` identifies the event to be deregistered
Return Value	`MDMI_NO_ERROR` on success otherwise an error

5.2.9 MdmiGetSessionStats

This function gets the statistics of the session.

Function Name	**`MdmiGetSessionStats`**
Signature	`int MdmiGetSessionStats(in MdmiSession session, out MdmiSessionStats stats);`
Arguments	`session:` identifies the session `stats:` the statistics to be returned
Return Value	`MDMI_NO_ERROR on success otherwise an error`

5.3 MIB

The MIB is modelled on SNMPv1 (see section 1.5 References). It provides a set of objects which can be retrieved and set through the MDMI. It does, however, diverge from SNMPv1.

6 Security

This section defines the Security Architecture Design for MDMI.

The operator owned server manages both authentication and log sessions in devices. User is authenticated prior to a session initiated by an on board application. In the case of remote session management, a command token is sent from server to the device.

The token is passed to the component being logged, by the diagnostic application, where the integrity of the token is verified and then parsed.

Selective logging, based on log mask setting in the command token, is enabled. Logs are collected by the diagnostic application, prior to sending them to the operator server.

All transmissions to / from the server shall use TLS version 1.2 or higher to ensure security of the user authentication information, command tokens, and log data while in transit. If the operating system provides for certificate revocation / update check, this should be used.

Note: The MDMI API for log session control functions, such as the log mask setting, is used for the development phase of the device and diagnostic application.

6. **Operating System Restrictions and Policy enforcement –** Permissions, Group ID enforcement, package Name, UID, package signature along with policy enforcement using SeLinux is incorporated in devices. This is to ensure only operator authorized on board log applications have access to the MDMI interface. Specification of group ID, permissions and policies are operator specific.
7. **User Authentication** – Server validates user with login credentials (user name and password), as supplied by user launching the application, thereby restricting only authenticated users to log into the diagnostic system. Additional checks based on IMEI, MSISDN, IMSI and user's group are used to restrict devices log and reports to specific users or user group.
8. **Secure Channel** – Command tokens are sent to the device from the server via a secure channel. As noted above, the secure channel shall use TLS version 1.2 or higher, and certificate revocation / update check shall be used if available.
9. **Command Set for remote log session** – JSON packets are used to define command tokens (ON/OFF, Time to Live, Log Privilege level, Log groups). JSON Packets are integrity protected and can be interpreted only by the modem (which has the public key of policy certificate).

Log Data – The log mask limits the scope of the logging information. The log mask can be set by a user or privileges enforced by the SW load. If log data cannot be transmitted to the server immediately, it shall be stored securely on the device, in a manner such that untrusted applications are unable to access the log data.

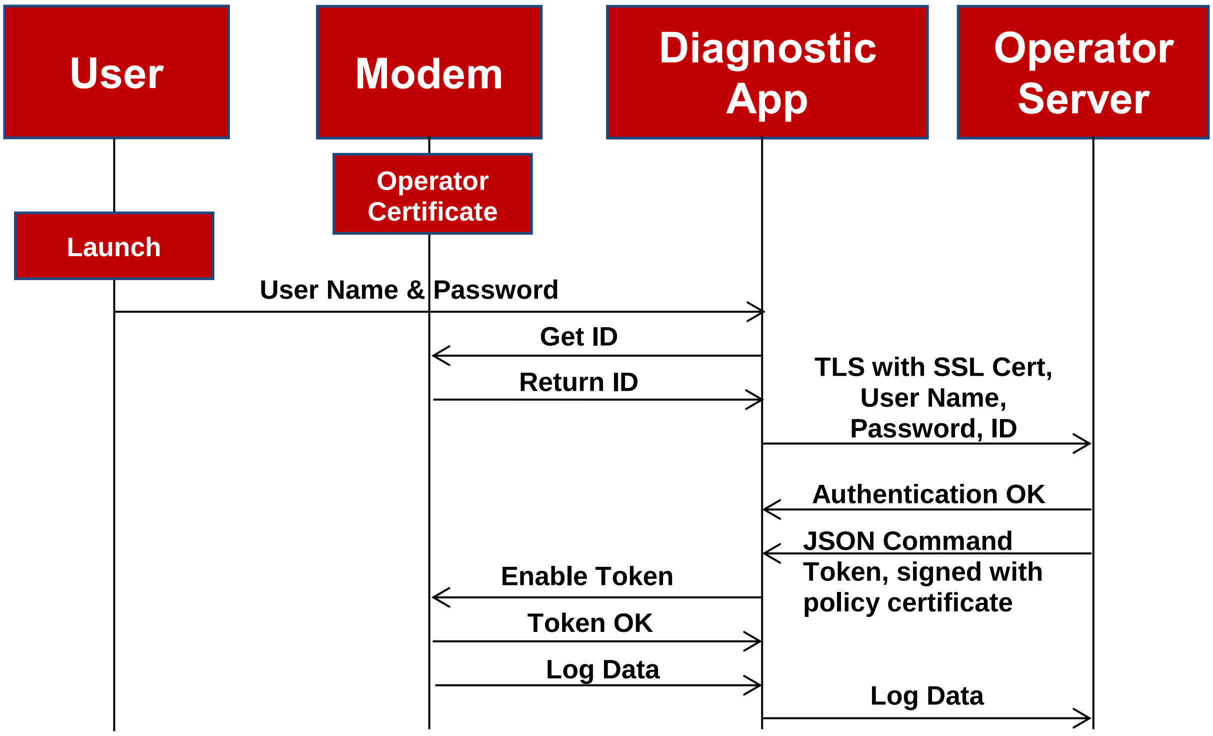


Figure 5 Security Call Flow

The logs contain the privacy information of device hardware and software thus all transmissions to / from the server shall use TLS version 1.2 or higher to ensure security. Besides, it is recommended to perform authentication between device modem, diagnosis platform and operator's server when the device is sending modem logs to the server. Figure 5 shows the device modem logs transmission authentication process.

(Note: this process is applicable for engineering device log transmission not for the commercial device)

1. Launch the diagnostic application platform on device by sending the user's name and password.

2. The diagnostic application platform allocates and stores IDs based on the target device modem log to be collected. Then diagnostic application platform sends IDs to different device modems. For example, the diagnostic application platform allocates ID1: 0001 for logs from device A modem and allocates ID2: 0002 for logs from device B modem.

3. The device will then store the received ID that allocated for its modem log. For example, device A modem will store ID1 and device B modem will store ID2.

4. If the logs of the target device modem (e.g. A) are required, the device will send the stored ID (e.g., 0001) to the diagnostic APP.

5. After verifying the ID send from modem is matched with the allocated ID (e.g. 00001). Diagnostic APP will send the User Name, Password, allocated ID to the server. Server will launch user authentication and then based on the information above produce the related token for this user.

6. A command token is sent from server to the diagnostic application (the token is signed with policy certificate), then device will receive the token that sent from the diagnostic application.

7. Since the key of policy certificate is stored within the modem, the integrity of the token is verified and then parsed on device: The device will decrypt the token and obtain the corresponding ID.

8. The device then verify that the Decrypted ID is matched with the stored ID (e.g. 0001), then respone the “Token OK” to diagnostic platform.

9. The logging modem on device will enable the related logging. Device will send the modem logs to the Diagnostic APP.

10. The Diagnostic APP transfers the modem logs to the Operator Server by the secure channel.

6.1 Example of a JSON Command Token

An example of a command token sent from operator server to the device, is provided below:

- "token_id", identifier for a token for the operator server
- "session_id", identifier for a particular log session
- "device_id", IMEI of the device
- "validity", length of log session
- "diag_cmd", diagnostic command
- "diag_mask", diagnostic log mask, specify what needs to be logged. Example, all RRC messages.

```
{
"diag_token":["token_id":"0xaf010230405", "session_id" : "0x1234435",
"device_id":"990000862471854","validity" : "8/6/2015 5:00:00PM EST",
"diag_cmd":"start_log", "diag_mask":"dnp3cm9ja3M="],
"signature":"OGI0ZDM4NTY0MzMyYTVmYWI2OGRhNTMzMmJmNWY5MThhZjU5ZWViNQ
=="
}
```

6.1.1 Steps for Building the Token

Although JSON does not mandate an order of the elements, an order is specified below, to enable correct HASH functions.

Server builds Hash of the "diag_token" array, for integrity protection, using the steps specified below:

10. Starting Values of the Token, SERVER_PUBLIC_KEY and SERVER_PRIVATE_KEY are defined.

Note: The SERVER_PUBLIC_KEY is contained in the component being logged and within the corresponding Certificate.

{"diag_token":["token_id":"0xaf010230405", "session_id" : "0x1234435", "device_id":"990000862471854","validity" : "8/6/2015 5:00:00PM EST", "diag_cmd":"start_log", "diag_mask":"dnp3cm9ja3M="], "signature_algorithm":"sha256RSA", "signature":"dRMxJCdBtMx/9q8RMiH8/719SB9roDNimYCdt43vp/7d3IEVuaj65aoYni+rwyMl

wmRXOJ3aqXJ4cxMGWJsJOSeKg/bcWlnHeDowPhoBxY3rj661kBI67QgDuI8X2KqCTMpI3
2hcGARJG0Xd4XyQdPLYTOmElPwm9a7Ckc3sOuM03dQoIqbs802HP8P0XWX/QyEOpZ2n
9yib6XIQSMzRSl+gM36PAQO8Fz/q/pUyBZOL7Mvnne9nOyssh7TJVLXcKkDwElKf3zr8CJ8
nCLY8kPhi5EqaW3zq/SlKo7GRHjBFDljoc2ke568QlxejG20mI2VYrw6wqaPCgdHs1k3Wmw
=="}

11. Extract all JSON Keys and Values *only* under "diag_token"

    "token_id":"0xaf010230405"

    "session_id" : "0x1234435"

    "device_id":"990000862471854"

    "validity" : "8/6/2015 5:00:00PM EST"

    "diag_cmd":"start_log"

    "diag_mask":"dnp3cm9ja3M="

12. Sort JSON Keys Pairs alphabetically, using ASCII UTF-8 characters (no special characters). Numbers get sorted in increasing trend.

    1: "device_id":"990000862471854"

    2: "diag_cmd":"start_log"

    3: "diag_mask":"dnp3cm9ja3M="

    4: "session_id" : "0x1234435"

    5: "token_id":"0xaf010230405"

    6: "validity" : "8/6/2015 5:00:00PM EST"

13. Concatenate All Key and Values without any delimiters. ASCII representation is used with UTF-8 characters. The output herein is called "Value Key Pair String"

    device_id990000862471854diag_cmdstart_logdiag_maskdnp3cm9ja3M=session_id0x1234435token_id0xaf010230405validity8/6/2015 5:00:00PM EST

14. Execute HASH over the "Value Key Pair String"

SHA256(device_id990000862471854diag_cmdstart_logdiag_maskdnp3cm9ja3M=session_id0x1234435token_id0xaf010230405validity8/6/2015 5:00:00PM EST) =
b78340828893b65963ed5777f138c4f930cb59dc2b85a2077c54bc1b90de3539

15. Sign the Hash value with the corresponding Private Key of Server, SERVER_PRIVATE_KEY

    Signature shall be done using binary value, and not hex

SIGN(b78340828893b65963ed5777f138c4f930cb59dc2b85a2077c54bc1b90de3539) = signature.bin

16. Base64 Encode Signature & Insert into JSON Token

{"diag_token":["token_id":"0xaf010230405", "session_id" : "0x1234435", "device_id":"990000862471854","validity" : "8/6/2015 5:00:00PM EST", "diag_cmd":"start_log", "diag_mask":"dnp3cm9ja3M="], "signature_algorithm":"sha256RSA", "signature":"dRMxJCdBtMx/9q8RMiH8/719SB9roDNimYCdt43vp/7d3IEVuaj65aoYni+rwyMl
wmRXOJ3aqXJ4cxMGWJsJOSeKg/bcWlnHeDowPhoBxY3rj661kBI67QgDuI8X2KqCTMpI3
2hcGARJG0Xd4XyQdPLYTOmElPwm9a7Ckc3sOuM03dQoIqbs802HP8P0XWX/QyEOpZ2n
9yib6XIQSMzRSl+gM36PAQO8Fz/q/pUyBZOL7Mvnne9nOyssh7TJVLXcKkDwElKf3zr8CJ8
nCLY8kPhi5EqaW3zq/SlKo7GRHjBFDljoc2ke568QlxejG20mI2VYrw6wqaPCgdHs1k3Wmw
=="}

6.1.2 Steps for Parsing the Token

The diagnostic application determines target component, by using a token ID. The component that receives the token shall use the following method to parse the token.

1. Obtain the value, ensure that the fields “signature_algorithm”, “signature” are populated and validate for format conformance (i.e. length)

{"diag_token":["token_id":"0xaf010230405", "session_id" : "0x1234435", "device_id":"990000862471854","validity" : "8/6/2015 5:00:00PM EST", "diag_cmd":"start_log", "diag_mask":"dnp3cm9ja3M="], "signature_algorithm":"sha256RSA", "signature":"dRMxJCdBtMx/9q8RMiH8/719SB9roDNimYCdt43vp/7d3IEVuaj65aoYni+rwyMl
wmRXOJ3aqXJ4cxMGWJsJOSeKg/bcWlnHeDowPhoBxY3rj661kBI67QgDuI8X2KqCTMpI3
2hcGARJG0Xd4XyQdPLYTOmElPwm9a7Ckc3sOuM03dQoIqbs802HP8P0XWX/QyEOpZ2n
9yib6XIQSMzRSl+gM36PAQO8Fz/q/pUyBZOL7Mvnne9nOyssh7TJVLXcKkDwElKf3zr8CJ8
nCLY8kPhi5EqaW3zq/SlKo7GRHjBFDljoc2ke568QlxejG20mI2VYrw6wqaPCgdHs1k3Wmw
=="}

2. Extract all JSON Keys and Values under "diag_token" only

"token_id":"0xaf010230405"

"session_id" : "0x1234435"

"device_id":"990000862471854"

"validity" : "8/6/2015 5:00:00PM EST"

"diag_cmd":"start_log"

"diag_mask":"dnp3cm9ja3M="

3. Sort JSON Keys Pairs alphabetically, using ASCII UTF-8 characters (no special characters). Numbers get sorted in increasing trend..

1: "device_id":"990000862471854"

2: "diag_cmd":"start_log"

3: "diag_mask":"dnp3cm9ja3M="

4: "session_id" : "0x1234435"

5: "token_id":"0xaf010230405"

6: "validity" : "8/6/2015 5:00:00PM EST"

4. Concatenate All Key and Values without any delimiters. ASCII representation is used with UTF-8 characters. The output herein is called "Value Key Pair String" device_id990000862471854diag_cmdstart_logdiag_maskdnp3cm9ja3M=session_id0x1234435token_id0xaf010230405validity8/6/2015 5:00:00PM EST

5. Execute HASH Over the "Value Key Pair String". to obtain **HASH_A.** The hash algorithm is specified in “signature_algorithm”

SHA256(device_id990000862471854diag_cmdstart_logdiag_maskdnp3cm9ja3M=session_id0x1234435token_id0xaf010230405validity8/6/2015 5:00:00PM EST) = **b78340828893b65963ed5777f138c4f930cb59dc2b85a2077c54bc1b90de3539 = HASH_A**

6. Extract “signature” field from JSON token, it should be Base64 encoded. Perform a Base64 Decode.

BASE64_DECODE(dRMxJCdBtMx/9q8RMiH8/719SB9roDNimYCdt43vp/7d3IEVuaj65aoYni+rwyMl

wmRXOJ3aqXJ4cxMGWJsJOSeKg/bcWlnHeDowPhoBxY3rj661kBI67QgDuI8X2KqCTMpI32hcGARJG0Xd4XyQdPLYTOmElPwm9a7Ckc3sOuM03dQoIqbs802HP8P0XWX/QyEOpZ2n9yib6XIQSMzRSl+gM36PAQO8Fz/q/pUyBZOL7Mvnne9nOyssh7TJVLXcKkDwElKf3zr8CJ8nCLY8kPhi5EqaW3zq/SlKo7GRHjBFDljoc2ke568QlxejG20mI2VYrw6wqaPCgdHs1k3Wmw==") = **SIGNATURE_RAW_BYTE_VALUE_HMAC**

7. Utilize the Public Key (SERVER_PUBLIC_KEY), to decrypt output from Step #5 (**SIGNATURE_RAW_BYTE_VALUE_HMAC**) This will yield **HASH_B**

8. If HASH_B = HASH_A, then signature verification is successful. Token signature is valid.

   Else abort the process and declare that signature verification failed.

9. Perform any additional checks per token policy (i.e. validity period, device_id, etc.).

10. If steps #7 and #8 are successful, integrity of token is verified and the logging component is ready for parsing commands and enabling related logging

Note: The Hashing algorithm and asymmetric encryption algorithm can be defined by the network operator i.e. the implementer. The implementer can enhance the steps by using industry standard algorithms such as HMAC.

SHA-1 has known weaknesses and shall not be used.

6.2 Log Security

6.2.1 Introduction

At this time, since the scope of the logging per this document is restricted to engineering builds end-to-end, encryption between the modem and the diagnostic app is not considered. In this initial phase, the log mask can be used to control the content of the log via user opt in/consent and privileges enforced by the SW load to address consumer privacy concerns. If commercial builds are supported in the future, encryption will need to be revisited at that time to prevent the log content from getting to unauthorized users.

6.2.2 Log Mask Management

The user/consumer payload, such as voice and emails, are most sensitive due to privacy issues. The best method to manage privacy concerns is an a priori setting, before logging starts. Log mask can be set such that all user payload (IP/UDP/TCP/RTP) are not logged, only the header information of these protocols will be logged.

Note: For SIP and RTCP, if packet logging is turned on, both header and payload shall be logged. This is because the payload of SIP and RTCP includes session options which are used for debug and analysis. This should not be a concern since consumer data is not available in SIP and RTCP payloads.

The log mask setting can be managed by an application configuration/consent where a user is provided an option/interface to select what data can be logged. The log mask can also be enforced with the SW load. For example, the operator field engineers may receive a load where all log masks are available, but if 3rd party vendors are testing for the operator, they may get a load where certain log masks cannot be enabled.

The log mask choices can be either packet header only logging, or packet payload logging. If the user opts for packet header only logging, the user can further select any combination of the following:

1. IP header,
17. UDP header,
18. TCP header,
19. SIP header and payload,
20. RTP header,
21. RTCP header and payload

If the user opts for packet payload logging, the user can further select any combination of the following:

2. IP header and payload,
3. UDP header and payload,
4. TCP header and payload,
5. SIP header and payload,
6. RTP header and payload,
7. RTCP header and payload

7 MDMI Implementation

Although MDMI is modelled on SNMP, there are several exceptions:

- The MDMI MIB does not fully follow the SNMP standard.
- The MDMI messaging interface uses a log record structure that follows an expanded version of the SNMP message standard.

7.1 MDMI MIB

- The MDMI OIDs do follow the hierarchical structuring rules defined in SNMP/MIB-2.[1]
- Some MDMI OIDs correspond to data types not allowed in SNMP/MIB-2, although allowed by ASN.1 syntax. Examples are
  - Boolean
  - Special data types – these are objects with multiple values. The objects have an OID, but the individual values do not.
    An example is ServingCellMeasurement (with its own OID), which includes the variables PCI, RSRP, RSRQ, RSSI, and SINR (none of which have an OID).

Because of these deviations from SNMP/MIB-2, implementations of the MDMI spec should not be based on the use of standard SNMP tools for generating the structures of the MDMI OIDs from the MDMI MIB file.

7.2 MDMI Log Record Structure

The interface between the DM application and the MDMI.so library follows a simplified and expanded version of the SNMP messaging format. Specifically, the message is implemented as an “MDMIValue” structure. The “data” element in that structure is a concatenation of “Length,” “Timestamp,” and “MDMI Message” as indicated in the following table.

Field	Length (bytes)	Description
Length	4	The length of the entire MDMI log record, in bytes, including Length itself, Timestamp and MDMI Message. For example, if the length of the MDMI log record is 10 bytes, the Length field should be [0x00, 0x00, 0x00, 0x0A].
Timestamp	8	The modem timestamp when the log record is constructed, given as the number of milliseconds since the January 1, 2015, 00:00:00 GMT epoch. For example, if the time is January 1, 2015, 01:00:00 GMT, which is 3,600,000 ms past the epoch, the Timestamp field should be [0x00, 0x00, 0x00, 0x00, 0x00, 0x36, 0xEE, 0x80]
MDMI Message	varies	The MDMI message, constructed according to the MDMI MIB. This includes the exceptions indicated in Section 7.1.The MDMI message, constructed according to the MDMI MIB. This includes the exceptions indicated in Section 7.1. Please refer to the Implementation Guideline for details.

[1] RFC1213, http://www.ietf.org/rfc/rfc1213.txt

7.2.1 MDMI Message

An MDMI message consists of a header indicating the PDU type followed by an (OID, value) pair.

There are three types of MDMI calls, each with a corresponding PDU type for a corresponding SNMP Message:

1. MdmiGet(), with corresponding PDU Type = GetResponse
22. MdmiSubscribe(), with corresponding PDU Type = Trap
23. MdmiInvoke(), with corresponding PDU Type = SetRequest

Figure 6 shows an example of the MDMI log record used with MdmiGet(). The particular call is for “deviceName,” which has an OID of 1.1.1.1, and the corresponding value is “MDMI_TEST_DEVICE.” The call is made at January 1, 2015, 01:00:00 GMT, which is 3,600,000 ms past the epoch.

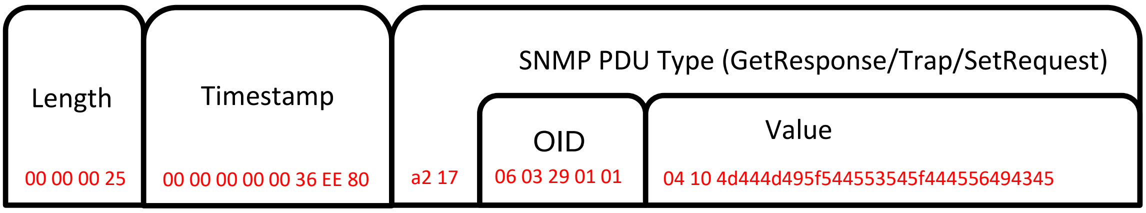

Length
00 00 00 25
Timestamp
00 00 00 00 00 36 EE 80
SNMP PDU Type (GetResponse/Trap/SetRequest)
a2 17
OID
06 03 29 01 01
Value
04 10 4d444d495f544553545f444556494345

Figure 6 Log Record for GetResponse / deviceName

The constituent fields are

- The Length is 4 + 8 + 25 = 37 = 0x25
- The Timestamp is 3600000 = 0x36EE80
- The PDU Type is encoded as a2 17
- The OID is encoded as 06 03 29 01 01.
    - The first byte 06 is the type of Object Identifier.
    - The second byte 03 is the length of the following Data field. (ASN.1 BER specifies how to encode lengths greater than 255.)
    - The remaining 3 bytes specify the OID, following ASN.1 BER
  - The first two numbers of any OID (x.y) are encoded as one value, using the formula (40*x) + y. Therefore, the first two numbers of the OID are encoded as (40*1) + 1 = 41 = 0x29.
  - The subsequent numbers in the OID are each encoded as one byte per number.
- The Value is encoded as 04 10 4d 44 4d 49 5f 54 45 53 54 5f 44 45 56 49 43 45.
    - The first byte 04 is the type of Octet String.

- The second byte 10 is the length of the following Data field, which is 16.
- The remaining 16 bytes form the octet string of “MDMI_TEST_DEVICE”.

Annex A UICC/eUICC Whitelist

The following symbols are used in the tables below to define the UICC/eUICC white list.

- ✗ -- indicates that the file cannot be traced
- ! -- indicates that only part of the file can be traced
- ✓ -- indicates that the entire file can be traced

Files under USIM

The following tables describe all the files present in the USIM application.

EFs with the UPDATE condition is not as administrative access (ADM), are described in the following slides. EFs that cannot be updated by the terminal are not covered.

These EFs normally contain static information, which are not sensitive, as it neither has user data nor any secret keys.

File name	ID	Content	
EF_{LI} (Language Indication)	6F05	Information about the language of the user	✗
EF_{Keys} (Ciphering and Integrity Keys)	6F08	Ciphering key CK, integrity key IK, and key set identifier KSI	✗
EF_{Keys} (Ciphering and Integrity Keys)	6F08	Ciphering key CK, integrity key IK, and key set identifier KSI	✗
EF_{KeysPS} (Ciphering and Integrity Keys for Packet Switched domain)	6F09	Ciphering key CKPS, integrity key IKPS, and key set identifier KSIPS for the packet switched (PS) domain	✗
$EF_{PLMNwAcT}$ (User controlled PLMN selector with Access Technology)	6F60	Defines the preferred PLMNs of the user in priority order	✓
EF_{ACMmax} (ACM maximum value)	6F37	Maximum value of the accumulated call meter	✓
EF_{ACM} (Accumulated Call Meter)	6F39	Total number of units for both current call and preceding calls	✓
EF_{PUCT} (Price per Unit and Currency Table)	6F41	Price per Unit and Currency Table (PUCT), which may be used to compute the cost of calls	✓
EF_{CBMI} (Cell Broadcast Message identifier selection)	6F45	The type of content of the cell broadcast messages that the subscriber wants the UE to accept	✗

File name	ID	Content	
EF_{FPLMN} (Forbidden PLMNs)	6F7B	List of forbidden PLMNs	✓
EF_{LOCI} (Location Information)	6F7E	Contains Temporary Mobile Subscriber Identity (TMSI), Location Area Information (LAI) and Location update status.	! TMSI & LAI values must be masked
EF_{CBMIR} (Cell Broadcast Message Identifier Range selection)	6F50	Ranges of cell broadcast message identifiers that the subscriber wants the UE to accept	✗
EF_{PSLOCI} (Packet Switched location information)	6F73	Contains Packet Temporary Mobile Subscriber Identity (P-TMSI), P-TMSI signature value, Routing Area Information (RAI), and Routing Area update status	✗
EF_{FDN} (Fixed Dialling Numbers)	6F3B	Contains Fixed Dialling Numbers (FDN)	! At least alpha and dialling number must be masked for privacy
EF_{SMS} (Short messages)	6F3C	Short messages (and parameters) which have been received or are to be sent	✗
EF_{MSISDN} (MSISDN)	6F40	MSISDN(s) related to the subscriber	✓
EF_{SMSP} (Short message service parameters)	6F42	Short Message Service header Parameters, such as service centre address	✓
EF_{SMSS} (SMS status)	6F43	Status information relating to the short message service (that is, Memory Capacity Exceeded)	✓
EF_{EXT2} (Extension2)	6F4B	Extension data of an FDN	! Only extension data must be masked (similar to EF_{FDN})
EF_{SMSR} (Short message status reports)	6F47	Short message status reports, which are received by the UE from the network	✓
EF_{ICI} (Incoming Call Information)	6F80	Time of the call and duration of the incoming calls	✗
EF_{OCI} (Outgoing Call Information)	6F81	Time of the call and duration of the outgoing calls	✗

File name	ID	Content	
EF_{ICT} (Incoming Call Timer)	6F82	Accumulated incoming call timer duration	✓
EF_{OCT} (Outgoing Call Timer)	6F83	Accumulated outgoing call timer duration	✓
EF_{EXT5} (Extension5)	6F4E	Extension data of EF_{ICI}, EFOCI and EF_{MSISDN}	✗
EF_{CCP2} (Capability Configuration Parameters 2)	6F4F	Parameters of required network and bearer capabilities and terminal configurations associated with a call established	✓
EF_{AaeM} (Automatic Answer for eMLPP Service)	6FB6	Priority levels for which the ME shall answer automatically to incoming calls	✓
$EF_{Hiddenkey}$ (Key for hidden phone book entries)	6FC3	Hidden key to display the phone book entries that are marked as hidden	✗
EF_{BDN} (Barred Dialling Numbers)	6F4D	Barred Dialling Numbers (BDN)	! At least alpha and dialling number must be masked for privacy.
EF_{EXT4} (Extension4)	6F55	Extension data of a BDN	! Only extension data must be masked.
EF_{EST} (Enabled Services Table)	6F56	List of enabled services	✓
EF_{ACL} (Access Point Name Control List)	6F57	List of allowed APNs (Access Point Names)	✗
EF_{DCK} (Depersonalisation Control Keys)	6F2C	Storage for the depersonalization control keys associated with the OTA depersonalization cycle	✓
$EF_{START-HFN}$ (Initialization values for Hyperframe number)	6F5B	Values of $START_{CS}$ and $START_{PS}$ of the bearers that were protected by the keys in EF_{KEYS} or EF_{KEYSPS} at release of the last CS or PS RRC connection	✗
EFNETPAR (Network Parameters)	6FC4	Information concerning the cell frequencies	✓
EFMBDN (Mailbox Dialling Numbers)	6FC7	Dialling numbers to access mailboxes	✓

File name	ID	Content	
EF$_{\text{EXT6}}$ (Extension6)	6FC8	Extension data of an MBDN	✓
EF$_{\text{MBI}}$ (Mailbox Identifier)	6FC9	Information to associate mailbox dialling numbers in EF$_{\text{MBDN}}$ with a message waiting indication group type and subscriber profile	✓
EF$_{\text{MWIS}}$ (Message Waiting Indication Status)	6FCA	Status of indicators that define whether a Voicemail, Fax, Electronic Mail, Other or Videomail message is waiting	✓
EF$_{\text{CFIS}}$ (Call Forwarding Indication Status)	6FCB	Status of indicators that are used to record whether call forward is active	✓
EF$_{\text{EXT7}}$ (Extension7)	6FCC	Extension data of a CFIS	✓
EF$_{\text{MMSN}}$ (MMS Notification)	6FCE	MMS notifications and parameters received by the UE	✓
EF$_{\text{EXT8}}$ (Extension 8)	6FCF	Extension data of an MMS Notification	✓
EF$_{\text{MMSUP}}$ (MMS User Preferences)	6FD1	Multimedia Messaging Service User Preferences, which can be used by the ME for user assistance in preparation of mobile multimedia messages	✗
EF$_{\text{MMSUCP}}$ (MMS User Connectivity Parameters)	6FD2	Values for Multimedia Messaging Connectivity Parameters used by the ME for MMS network connection	✓
EF$_{\textbf{VGCSS}}$ (Voice Group Call Service Status)	6FB2	Status of activation for the VGCS group identifiers	✓
EF$_{\textbf{VBSS}}$ (Voice Broadcast Service Status)	6FB4	Status of activation for the VBS group identifiers	✓
EF$_{\text{GBABP}}$ (GBA Bootstrapping parameters)	6FD6	AKA Random challenge (RAND) and Bootstrapping Transaction Identifier (B-TID) associated with a GBA bootstrapping procedure	✗
EF$_{\text{EPSLOCI}}$ (EPS location information)	6FE3	Contains Globally Unique Temporary Identifier (GUTI), Last visited registered Tracking Area Identity (TAI) and EPS update status.	! GUTI & TAI must be masked
EF$_{\text{EPSNSC}}$ (EPS NAS Security Context)	6FE4	EPS NAS Security context	✗
EF$_{\text{FDNURI}}$ (Fixed Dialling Numbers URI)	6FED	List of FDN stored in URI address format	✗
EF$_{\text{BDNURI}}$ (Barred Dialling Numbers URI)	6FEE	List of BDN stored in URI address format	✗

Files under USIM/DF Phonebook

Many files in the DF-Phonebook contain sensitive data from privacy perspective, hence they should not be traced.

Many files do not have a fixed file ID, but this is derived based on the parsing of the EF-PBR. To avoid issues during the tracing, it is recommended to avoid logging all EFs that have a non-fixed file ID.

The above list only includes EFs that can be updated by the user and have a fixed file ID.

File name	ID	Content	
EF$_{PSC}$ (Phone book Synchronization Counter)	6F22	Used to construct the phone book identifier (PBID) to determine whether the accessed phone book is the same as previous access	✗
EF$_{CC}$ (Change Counter)	6F23	Used to detect changes made to the phone book	✗
EF$_{PUID}$ (Previous Unique Identifier)	6F24	Previously used unique identifier (UID)	✗

Files under USIM/DF GSM-ACCESS

File name	ID	Content	
EF$_{Kc}$ (GSM Ciphering key Kc)	4F20	Ciphering key Kc and the ciphering key sequence number n for enciphering in a GSM access network	✗
EF$_{KcGPRS}$ (GPRS Ciphering key KcGPRS)	4F52	Ciphering key KcGPRS and the ciphering key sequence number n for GPRS	✗
EF$_{CPBCCH}$ (CPBCCH Information)	4F63	Information concerning the CPBCCH to reduce the search of CPBCCH carriers when selecting a cell	✓

Files under USIM/DF WLAN

File name	ID	Content	
EF$_{Pseudo}$ (Pseudonym)	4F41	Temporary user identifier (pseudonym) for subscriber identification	! Pseudonym length is ok, but Pseudonym must be masked
EF$_{UPLMNWLAN}$ (User controlled PLMN selector for I-WLAN Access)	4F42	Preferred PLMNs to be used for WLAN PLMN Selection	✓

File name	ID	Content	
EF_{UWSIDL} (User controlled WLAN Specific Identifier List)	4F44	User preferred list of WLAN-specific identifier (WSID) for WLAN selection in priority order	✓
EF_{WRI} (WLAN Re-authentication Identity)	4F46	List of parameters linked to a re-authentication identity to be used in fast re-authentication	✗
$EF_{WLRPLMN}$ (I-WLAN Last Registered PLMN)	4F4A	I-WLAN Last Registered PLMN Selection	✓

Files under USIM/DF HNB

File name	ID	Content	
EF_{ACSGL} (Allowed CSG Lists)	4F81	CSG ID, HNB name, and CSG type in allowed CSG lists controlled by user	✗
EF_{CSGT} (CSG Type)	4F82	CSG Type	✓
EF_{HNBN} (Home NodeB Name)	4F83	Home NodeB Name	✗

Files under USIM/DF ProSe

File name	ID	Content	
EF_{PROSE_GC} (ProSe Group Counter)	4F09	PTK ID and Counter associated with the PGK currently in use for a ProSe Group	✗

Files under USIM/Other DFs

The following DFs are present in the USIM application, but do not contain any EF that can be updated by the terminal:

- DF-SoLSA
- DF-MexE
- DF-ACDC

Files under TELECOM

The TELECOM DF contains many files for backward compatibility with 2G terminals. These EFs must not be accessible by a 3G device.

Those EFs can potentially contain sensitive information and must not be logged.

DF-PHONEBOOK is also present under TELECOM. The same rules discussed for the phonebook inside the USIM apply, as the structure is same.

The below table provides the analysis for the remaining EFs in the TELECOM DF:

File name	ID	Content	
EF$_{\text{ICE\_DN}}$ (In Case of Emergency – Dialling Number)	6FE0	Number formatted in-case-of-emergency information	! At least alpha and dialling number must be masked for privacy.
EF$_{\text{ICE\_FF}}$ (In Case of Emergency – Free Format)	6FE1	Free formatted in-case-of-emergency information	✗
EF$_{\text{PSISMSC}}$ (Public Service Identity of the SM-SC)	6FE5	Public Service Identity of the SM-SC for SMS over IP	✓
EF$_{\text{ICE\_graphics}}$ (In Case of Emergency – Graphics)	5F50 / 4F21	ICE graphical information	✗
EF$_{\textbf{MML}}$ (Multimedia Messages List)	5F3B / 4F47	MM data stored in EF$_{\textbf{MMDF}}$	✗
EF$_{\text{MMDF}}$ (Multimedia Messages Data File)	5F3B / 4F48	Multimedia Messages data	✗

UICC Commands for USIM

This table provides details about logging concerns for each command used by the terminal, while interacting with the USIM.

Command	Description	Analysis
STATUS	Used by the terminal for polling and to complete USIM initialization or start its termination.	✓
SELECT DEACTIVATE FILE ACTIVATE FILE SEARCH RECORD	These commands perform operations on EFs present in the UICC card, but without exposing the content of those files.	✓
READ BINARY UPDATE BINARY READ RECORD UPDATE RECORD INCREASE	These commands perform operations on EFs present in the UICC card, either writing or reading their content.	! Behaviour depends on the specific EF that is accessed.

Command	Description	Analysis
RETRIEVE DATA SET DATA		
VERIFY PIN CHANGE PIN DISABLE PIN ENABLE PIN UNBLOCK PIN	These commands perform various operations on the PIN of the UICC. The value of the PIN is clearly sensitive data and should not be logged. Anyway, other parts of the transaction, such as number of remaining attempts or length of the PIN can be logged.	! Only the PIN/PUK values must be masked, while the rest of the command can be logged.
MANAGE CHANNEL	Used to open or close logical channels	✓
TERMINAL CAPABILITY	Used to provide terminal capabilities to the UICC	✓
AUTHENTICATE	Authentication command	! See next table

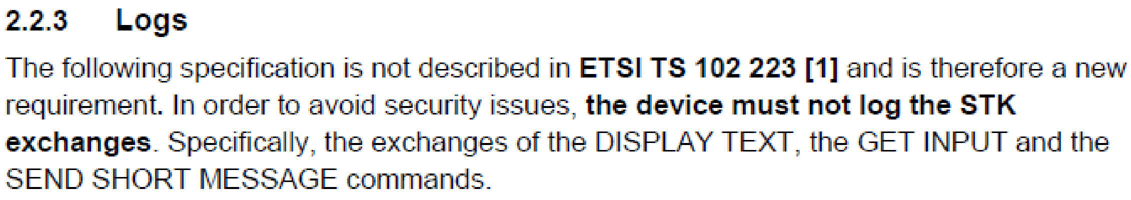
This Table provides details about logging concerns for the AUTHENTICATE command used by the terminal, while interacting with the USIM.

Author context	P2	Description	Analysis
GSM context	0x80	The command contains RAND. The response contains the SRES and Kc.	✗
3G context	0x81	The command contains RAND and AUTN. The response contains 1 byte for the result, followed by RES, CK, IK and optionally Kc. Alternatively, the response might contain the AUTS for resynchronization.	✗
VGCS/VBS context	0x82	The command contains Vservice_Id, VK_Id and VSTK_RAND. The response contains 1 byte for the result, followed by the VSTK.	✗
GBA context – Bootstrapping Mode	0x84	The command contains RAND and AUTN. The response contains 1 byte for the result, followed by RES or AUTS.	✗
GBA context – NAF Derivation Mode	0x84	The command contains NAF_ID and IMPI. The response contains 1 byte for the result, followed by Ks_ext_NAF.	✗
MBMS context	0x85	TBC	! TBC
Local Key Establishment mode	0x86	TBC	! TBC

Toolkit

It is difficult to classify the content of toolkit, as it depends on the applets running in the UICC. For example, the DISPLAY TEXT command could potentially be used to display sensitive data to the user.

There is a document published by GSMA (PDATA.12) with this requirement:



2.2.3 Logs

The following specification is not described in **ETSI TS 102 223 [1]** and is therefore a new requirement. In order to avoid security issues, **the device must not log the STK exchanges**. Specifically, the exchanges of the DISPLAY TEXT, the GET INPUT and the SEND SHORT MESSAGE commands.

The potential issues are not limited to those commands and the above requirement may be applicable to all STK exchanges (even if few commands are called out specifically).

Three ways are identified:

- Avoid logging all toolkit interactions.
- Allow logging of only some commands (for example, REFRESH) and mask other commands (for example, SEND SHORT MESSAGE).
- Peek into each TLV inside each command to log only those that are safe (for example, the Command details TLV inside the SEND SHORT MESSAGE is ok, while the 3GPP-SMS TPDU is not).

#	Approach	Pros and cons
1	Avoid logging all toolkit interactions	**Pros:** ▪ Very simple to implement. ▪ compliant with requirement in PDATA.12. **Cons:** ▪ Debugging of specific issues can become problematic, as much information exchanged between terminal and UICC is lost.
2	Allow logging of only some commands (for example, REFRESH) and mask other commands	**Pros:** ▪ Still simple to implement, as the type of proactive command or ENVELOPE can be identified easily. **Cons:** ▪ For some commands, the entire command is dropped from the logs, even if a large part of it does not contain sensitive information.
3	Peek into each TLV inside each command to log only those that are safe	**Pros:** ▪ Gives maximum control, allowing to only mask in the log fields that contain potentially sensitive data. **Cons:** ▪ Very complex to implement, as logic is required to decode the APDU right when it is received from the UICC.

Recommendation: Approach # 2 is a good compromise between the need to log toolkit interaction and the possibility to maintain low complexity.

All proactive commands and terminal responses have the same structure at the beginning. None of these contain sensitive information, while they can be useful for debugging.

Proactive command	Terminal response
Proactive UICC command Tag Length Command details Device Identities	Command details Device Identities Result

In case of both proactive commands and terminal responses, the specific type of command can be easily identified from the Command details. This can be extracted easily.

Proposal

- Extract the type of command from the “Command details”.
- Trace only the header (to be explicitly defined) for messages marked as ! in the next table.
- Trace the entire data for proactive commands marked as ✓ in the next table.

Proactive command	Usage	Command	Response
DISPLAY TEXT	Displays a text message, and/or an icon	! Command might contain sensitive data to be displayed to the user	✓
GET INKEY	Display text and/or an icon and expects the user to enter a single character.	! Command might contain sensitive data to be displayed to the user	! Response contains key from user, which can be sensitive.
GET INPUT	Displays text and/or an icon and any response string entered by the user is passed transparently to the UICC	! Command might contain sensitive data to be displayed to the user	! Response contains string typed by user, which can be sensitive.
MORE TIME	Allows the Card Application Toolkit (CAT) task in the UICC more time for processing	✓	✓
PLAY TONE	Plays an audio tone.	✓	✓
POLL INTERVAL	Requests how often the terminal shall send STATUS commands related to Proactive Polling	✓	✓
SET-UP MENU	Supplies a set of menu items to be integrated with the menu system	! Command might contain	✓

Proactive command	Usage	Command	Response
		sensitive data to be displayed to the user	
SELECT ITEM	Supplies a set of items from which the user may choose one.	! Command might contain sensitive data to be displayed to the user	✓
SEND SHORT MESSAGE	Requests the terminal to send a short message	! Command might contain sensitive data in the SMS payload	✓
SET UP CALL	Request to set up a voice call	! Command might contain sensitive data, such as phone number	✓
REFRESH	Notifies the terminal of the changes to the UICC configuration that have occurred as the result of a Network access application (NAA) application activity.	✓	✓
POLLING OFF	Disables the Proactive Polling	✓	✓
PROVIDE LOCAL INFORMATION	Requests the terminal to send current local information to the UICC	✓	! Response might contain sensitive data, such as IMEI or location.
SET UP EVENT LIST	Supply a set of events that UICC must receive	✓	✓
PERFORM CARD APDU	Requests the terminal to send an APDU command to the additional card	! Command might contain sensitive data in the APDU.	! Command might contain sensitive data in the APDU.
POWER OFF CARD	Requests the terminal to close a session with the additional card	✓	✓
POWER ON CARD	Requests the terminal to start a session with the additional card	✓	✓
GET READER STATUS	Requests the terminal to get information about all interfaces or the indicated interface to additional card reader(s).	✓	✓

Proactive command	Usage	Command	Response
TIMER MANAGEMENT	Manages timers running physically in the terminal	✓	✓
SET UP IDLE MODE TEXT	Supplies a text string to be displayed by the terminal as an idle mode text	! Command might contain sensitive data to be displayed to the user	✓
RUN AT COMMAND	Sends an AT Command to the terminal as though initiated by an attached TE	! Command might contain sensitive data.	! Response might contain sensitive data.
SEND DTMF COMMAND	Send a DTMF string after a call has been successfully established	! DTMF sequence can be a sensitive code	✓
LANGUAGE NOTIFICATION	Notifies the terminal about the language currently used for any text string	✓	✓
LAUNCH BROWSER	Requests a browser in the terminal to open a specified URL.	! URL might contain sensitive data	✓
OPEN CHANNEL	Opens a channel to transmit data	! Command might contain sensitive data such as server address	✓
CLOSE CHANNEL	Close a channel	✓	✓
RECEIVE DATA	Returns data from a channel	✓	! Response might contain sensitive data received over the channel.
SEND DATA	Sends data on a channel	! Command might contain sensitive data to be sent over the channel	✓
GET CHANNEL STATUS	Returns the channel status	✓	✓
SERVICE SEARCH	Searches for the availability of a service in the environment of the terminal	✓	✓
GET SERVICE INFORMATION	Looks for the complete service record related to a service	✓	✓

Proactive command	Usage	Command	Response
DECLARE SERVICE	Downloads into the terminal service database the services that the card provides as a server	✓	✓
SET FRAMES	Instructs the terminal to divide the screen into multiple rectangular regions	✓	✓
GET FRAMES STATUS	Returns status of the frames	✓	✓
RETRIEVE MULTIMEDIA MESSAGE	Instructs the terminal to retrieve a multimedia message	! Command might contain sensitive data	✓
SUBMIT MULTIMEDIA MESSAGE	Instructs the terminal to submit a multimedia message	! Command might contain sensitive data	✓
DISPLAY MULTIMEDIA MESSAGE	Displays a multimedia message	! Command might contain sensitive data	✓
ACTIVATE	Activates a specified interface	✓	✓
CONTACTLESS STATE CHANGED	Informs the terminal when the contactless functionality in the UICC has been enabled or disabled	✓	✓
COMMAND CONTAINER	Send a CAT command to an eCAT (encapsulated CAT) client by encapsulation	! Command might contain sensitive data.	! Response might contain sensitive data.
ENCAPSULATED SESSION CONTROL	Ends an encapsulated command session with an eCAT client	✓	✓

A separate analysis is required for ENVELOPE commands. The specific ENVELOPE type can be recognized by the first byte and the first byte is the only common part for all ENVELOPE commands.

Many ENVELOPE commands contain sensitive information, such as position, and phone numbers.

Proposal

- Extract the type of ENVELOPE from the first byte.
- Trace only the ENVELOPE type for messages marked as ! in the next table.
- Trace the entire data of the ENVELOPE for messages marked as ✓ in the next table.

ENVELOPE	Usage	Command	Response
MENU SELECTION	Indicates the menu selected by the user	! Command might contain sensitive data.	✓
CALL CONTROL	Allows the UICC to modify or reject an outgoing voice/SMS/data call	! Command might contain sensitive data.	! Response might contain sensitive data.
TIMER EXPIRATION	Indicates expiration of a timer started by the UICC	✓	✓
EVENT DOWNLOAD	Used to communicate event to the UICC.	! Depends on specific event, but often contains sensitive data	✓
MMS TRANSFER STATUS	Informs the UICC about transfer of MMS	✓	✓
MMS NOTIFICATION DOWNLOAD	Provides MMS notification to the UICC	! Command might contain sensitive data	✓
TERMINAL APPLICATIONS	Provides list of terminal applications	✓	✓
ENVELOPE CONTAINER	Used to send ENVELOPE by eCAT client	! Command might contain sensitive data.	! Response might contain sensitive data.
SERVICE LIST	Provides a secure method for the terminal to retrieve CAT related information stored in the service tables of NAAs	! Command might contain sensitive data	✓
SMS-PP DATA DOWNLOAD	Allows the data download via SMS Point-to-point, often used for remote management of the UICC	! Command might contain sensitive data.	! Response might contain sensitive data.
USSD DATA DOWNLOAD	Allows the data download via USSD	! Command might contain sensitive data.	! Response might contain sensitive data.
Geographical Location Reporting	Reports the GPS location to the UICC	! Command might contain sensitive data.	✓
ProSe Report	Stores the ProSe report in the UICC	! Depends on specific	✓

ENVELOPE	Usage	Command	Response
		event, but often contains sensitive data.	

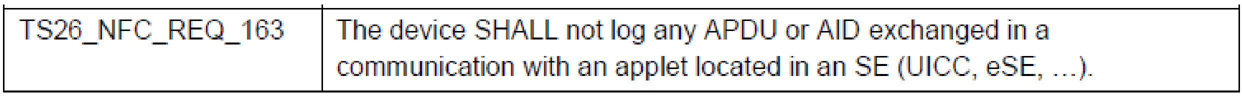
Non-telecom Channels

In recent years, the number of clients sending APDUs to the UICC for non-telecom applications has increased. Non-telecom applications refer to all applications that are not defined by 3GPP or 3GPP2.

SIM Alliance has standardized the Open Mobile APIs to allow a client to exchange APDUs with a UICC.

The modem has no information on what type of application is accessed and exchanged between the application on the device and the UICC.

GSMA TS.26 contains the following requirement.



TS26_NFC_REQ_163	The device SHALL not log any APDU or AID exchanged in a communication with an applet located in an SE (UICC, eSE, …).

To comply with GSMA requirements, masking is recommended for:

- the AID in all SELECT by DF name commands sent from the terminal to the UICC.
- all APDUs exchanged on logical channels where a non-telecom application is selected.
- The log can contain the CLA and INS byte, but it must not show anything else.

Annex B Source Code

Open source code is publicly available for vendors who are complying with the GSMA TSG standard diagnostic interface standard at GSMA site:

- https://github.com/GSMATerminals/TSG-Standard-Diag-Public.
- Summary of the files available are listed below.
- MDMI.h - header file specifying the MDMI interface.
- MDMI-MIB.txt - MIB file specifying log objects to be referenced in MDMI messages.
- Mdmi_sample_setgetinvoke.txt - example source code demonstrating MDMI usage for MdmiGet, MdmiSet, and MdmiInvoke.
- README.txt – MDMI Implementation guideline and revision history.
- Note: A user account must be created at the following site before access to the GitHub site.
- https://github.com/

Annex C Document Management

C.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
1.0	January 2016	New PRD TS.31	TSG#21 PSMC	Carol Becht / Verizon
2.0	April 2016	• Clarify use of SNMP • Description of MDMI Implementation and log record structure Add definition of 'Engineering build' • Add additional clarification and requirements on security	TSG	Carol Becht / Verizon
3.0	July 2016	• Update architecture to support multiple diagnostic feeds • Modify JSON token generation method to specify order of items (to enable correct HASHing)	TSG	Carol Becht / Verizon
4.0	November 2017	• Extend to UICC and eUICC logging • MDMI Library Discovery Database and MDMI Java Interface • For details see CR1004	TSG#30	Dhruv Khettry / Verizon
5.0	Sept 2021	Update with CR 1007 adding clarity to the flow diagram in Fig 5	TSG#45 ISAG#12	Paul Gosden / GSMA

Other Information

Type	Description
Document Owner	Terminal Steering Group
Editor / Company	Paul Gosden GSMA

It is our intention to provide a quality product for your use. If you find any errors or omissions, please contact us with your comments. You may notify us at prd@gsma.com

Your comments or suggestions & questions are always welcome.