OMA Device Management Protocol

Approved Version 2.0 – 09 Feb 2016

Open Mobile Alliance

OMA-TS-DM_Protocol-V2_0-20160209-A

Contents

Figures

Tables

1. Scope

This protocol is called the OMA Device Management Protocol version 2.0, and it defines the protocol for various management procedures. The scope for this protocol is to define the interfaces that are used between the DM Server and the DM Client. Interfaces residing within the device or within the server are outside of the scope of this specification.

2. References

2.1 Normative References

[3GPP23.003] "Numbering, addressing and identification", 3GPP, URL:http://www.3gpp.org

[3GPP31.102] "Characteristics of the Universal Subscriber Identity Module (USIM) application", 3GPP, URL:http://www.3gpp.org

[C.S0023-B_v1.0] "Removable User Identity Module For Spread Spectrum Systems", 3GPP2 C.S0023-B version 1.0, URL:http://www.3gpp2.org/Public_html/specs/C.S0023-B_v1.0_040426.pdf

[DM-AD] "Device Management Architecture", Open Mobile Alliance™, OMA-AD-DM-V2_0, URL:http://www.openmobilealliance.org/

[GCM] "Google Cloud Messaging for Android", Google, URL:http://developer.android.com/google/gcm/index.html

[GPSCP03] "GlobalPlatform Secure Channel Protocol 03 (SCP 03) Amendment D v1.1 Sept 2009", URL:http://www.globalplatform.org/specificationscard.asp

[ISO29115] Information technology- Security Techniques – Entity authentication assurance framework (2013-04-01), URL:http://www.iso.org

[ISO639-2] "Codes for the representation of names of languages-- Part 2: alpha-3 code" URL:http://www.loc.gov/standards/iso639-2/

[JSON-SCHEMA] "JSON Schema: core definitions and terminology, draft-zyp-json-schema-04", K. Zyp, Ed, January 31, 2013. URL:http://tools.ietf.org/html/draft-zyp-json-schema-04

[PKCS#15] "PKCS #15 v1.1: Cryptographic Token Information Syntax Standard", RSA Laboratories, June 6, 2000. URL:ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-15/pkcs-15v1_1.pdf

[POSIX] ISO/IEC/IEEE 9945-2009 Information Technology — Portable Operating System Interface (POSIX®) Base Specifications, Issue 7.

[PUSHOTA] "Push Over The Air", Open Mobile Alliance™, OMA_TS-PushOTA-V2_3, URL: http://www.openmobilealliance.org

[REST] "Architectural Styles and the Design of Network-based Software Architectures", Roy Thomas Fielding, 2000 URL:http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

[RFC1521] "MIME (Multipurpose Internet Mail Extensions) Part One", Borenstein & Freed, Sep 1993 URL:http://www.ietf.org/rfc/rfc1521.txt

[RFC1766] "Tags for the Identification of Languages". H. Alvestrand. March 1995. URL:http://www.ietf.org/rfc/rfc1766.txt

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997, URL:http://www.ietf.org/rfc/rfc2119.txt

[RFC2616] "Hypertext Transfer Protocol -- HTTP/1.1", R, Fielding, June 1999, URL:http://www.ietf.org/rfc/rfc2616.txt

[RFC2617] "HTTP Authentication: Basic and Digest Access Authentication", J. Franks, et al., June 1999, URL:http://www.ietf.org/rfc/rfc2617.txt

[RFC2817] "Upgrading to TLS Within HTTP/1.1", R. Khare and S. Lawrence, May 2000, URL:http://www.ietf.org/rfc/rfc2817.txt

[RFC2818] "HTTP over TLS", E. Rescorla, May 2000, URL:http://www.ietf.org/rfc/rfc2818.txt

[RFC3629] "UTF-8, a transformation format of ISO 10646", F. Yergeau, November 2003, URL:http://tools.ietf.org/html/rfc3629

[RFC3986] "Uniform Resource Identifier (URI): Generic Syntax", Berners-Lee, et al. Jan 2005, URL:http://www.ietf.org/rfc/rfc3986.txt

[RFC4122] "A Universally Unique Identifier (UUID) URN Namespace", P. Leach, et al. July 2005, URL:http://www.ietf.org/rfc/rfc4122.txt

[RFC4234] "Augmented BNF for Syntax Specifications: ABNF". D. Crocker, Ed., P. Overell. October 2005, URL:http://www.ietf.org/rfc/rfc4234.txt

[RFC4279] "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", P. Eronen, H. Tschofenig, December 2005, URL:http://www.ietf.org/rfc/rfc4279

[RFC4346] "The Transport Layer Security (TLS) Protocol Version 1.1", T. Dierks and E. Rescorla, April 2006, URL:http://www.ietf.org/rfc/rfc4346.txt

[RFC4627] "The application/json Media Type for JavaScript Object Notation (JSON)", D. Crockford. July 2006, URL:http://www.ietf.org/rfc/rfc4627.txt

[RFC5198] "Unicode Format for Network Interchange", J. Klensin, M. Padlipsky, March 2003, URL:http://www.ietf.org/rfc/rfc5198.txt

[RFC5246] "The Transport Layer Security (TLS) Protocol Version 1.2", T. Dierks and E. Rescorla, August 2008, URL:http://www.ietf.org/rfc/rfc5246.txt

[RFC5280] "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", D. Cooper, S. Santesson, S. Farrell, May 2008, URL:http://www.ietf.org/rfc/rfc5280.txt

[RFC5289] "TLS Elliptic Curve Cipher Suites with SHA-256/384 and AES Galois Counter Mode (GCM)", E. Rescorla, August 2008 URL:http://www.ietf.org/rfc/rfc5289.txt

[RFC6125] "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", P. Saint-Andre, J. Hodges, March 2011, URL:http://www.ietf.org/rfc/rfc6125.txt

[RFC6234] "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", D. Eastlake 3rd, etc. May 2011 URL:http://www.ietf.org/rfc/rfc6234.txt

[SCRRULES] "SCR Rules and Procedures", Open Mobile Alliance™, OMA-ORG-SCR_Rules_and_Procedures, URL:http://www.openmobilealliance.org/

[TS102.221] "Smart Cards; UICC-Terminal interface; Physical and logical characteristics", (ETSI TS 102 221 release 6), URL:http://www.etsi.org/

[TS151.011] "Specification of the Subscriber Identity Module - Mobile Equipment (SIM-ME) interface", (ETSI TS 151 011), URL:http://www.etsi.org/

[TS33.220] "Generic Bootstrapping Architecture (GBA)", URL:http://www.3gpp.org

[TS33.223] "Generic Bootstrapping Architecture (GBA) Push function", URL:http://www.3gpp.org

2.2 Informative References

[DM_1.3] "OMA Device Management Protocol", Version 1.3, Open Mobile Alliance™, URL:http://www.openmobilealliance.org/

[OMADICT] "Dictionary for OMA Specifications", Version 2.9, Open Mobile Alliance™, OMA-ORG-Dictionary-V2_9, URL:http://www.openmobilealliance.org/

[SSL] "The SSL 3.0 Protocol", A. Frier, P. Karlton, and P. Kocher, Nov 1996, URL:http://www.ietf.org/rfc/rfc6101.txt

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

3.2 Definitions

BOM	A “Byte Order Mark” consists of characters at the beginning of a data stream. It has no meaning for a Byte oriented data stream for which the encoding form is known (e.g. UTF-8 encoded data stream).
Cipher Suite	A cipher suite is a named combination of authentication, encryption, and message authentication code (MAC) algorithms used to negotiate the security settings for a network connection using the Transport Layer Security (TLS) / Secure Sockets Layer (SSL) network protocol.
DM Bootstrap	Message sent by Management Authority to the DM Client outside the context of a DM Session with the purpose of configuring the DM Client with data needed to connect to a DM Server
DM Command	Instruction which specifies tha atomic management operation.
DM Notification	Message sent from the DM Server to DM Client to alert DM Client to initiate a DM session back for management purpose.
DM Package	Unit of message transferred between the DM Client and the DM Server inside the context of a DM Session
DM Session	See Management Session.
DM Tree	The collection of the MO Instances exposed by the DM Client. See also Management Tree
JSON	The JSON refers to the definition of [RFC4627].
MO Instance	Management Object Instance is an occurrence of a Management Object exposed by the DM Client. Instances of a MO share the same node definitions and behaviours, and can be represented as a set of related nodes exposed by the DM Client. Multiple Instances of one MO MAY exist in the DM Tree.

Kindly consult [OMADICT] for more definitions used in this document.

3.3 Abbreviations

GCM	Google Cloud Messaging
HTTP	See [RFC2616]
MIID	MO Instance Identifier
MOS	MO Supported
REST	Representational state transfer [REST]
RESTful	REST compliant

Kindly consult [OMADICT] for all definitions used in this document.

4. Introduction (Informative)

Device Management refers to the management of device configuration and other managed objects of devices from the point of view of the DM Authorities. DM includes, but is not restricted to setting initial configuration information in devices, subsequent updates of persistent information in devices, retrieval of management information from devices, execute primitives on devices, and processing events and alarms generated by devices.

DM allows network operators, Service Providers or corporate information management departments to carry out the procedures of configuring devices on behalf of the End User (Customer).

4.1 Version 1.2

Device Management is the generic term used for technology that allows third parties to carry out the difficult procedures of configuring devices on behalf of the End User (Customer). Third parties would typically be operators, Service Providers or corporate information management departments.

Through DM, an external party can remotely set parameters, conduct troubleshooting servicing of terminals, install or upgrade software. In broad terms, DM consists of three parts:

- Protocol and mechanism: the protocol used between a management server and a device
- Data model: the data made available for remote manipulation, for example browser and mail settings
- Policy: the Policy decides who can manipulate a particular parameter, or update a particular object in the device

The specifications in the DM Enabler version 1.2 address the first part of DM above, the protocol and mechanism. More particularly, this Enabler release addresses the management of devices by specifying a protocol and management mechanisms that may be exposed by an OMA DM Client and targeted by an OMA DM Server.

The architecture of the DM Enabler anticipates the needs of the market actors to differentiate their products through vendor-specific extensions while providing a core parameter set that can be relied upon in all terminals exposing this standardized interface.

The design of the architecture follows the OMA architecture principle [ARCH-PRINC] of Network Technology Independence by separating the bearer-neutral requirements from bearer-specific bindings. The described architecture also anticipates additional bearer and proxy types, as any are identified, without requiring a respecification of previously released documents. This preserves vendor and customer investment while supporting the scaling required by future innovations.

There are three parts to the object schema that provide break-points between more general and more specific parameters:

- A top level Management Object which is bearer-neutral;
- A set of bearer-specific parameters;
- Sub-tree(s) for exposing vendor-specific parameters.

By composing the MOs in this way, it becomes possible for a DM Authority to:

- Target generic requirements that span all implementations;
- Focus on bearer-specific idiosyncrasies of a given networking environment;
- Activate terminal-specific behaviour by adjusting vendor-specific parameters.

In a wireless environment, the crucial element for DM Protocol is the need to efficiently and effectively address the characteristics of devices including low bandwidth and high latency and to provide for support of these management operations remotely, over-the-air.

4.2 Version 1.3

OMA DM Version 1.3 [DM_1.3] reused the architecture from OMA DM Version 1.2. It introduces new notification, transport protocols and a new DM Server to DM Server interface for delegation.

4.3 Version 2.0

OMA DM Version 2.0 reuses Management Objects which are designed for DM Version 1.3 or earlier DM Protocols. OMA DM Version 2.0 introduces new Client-Server DM Protocol, that is not backward compatible with any prior DM Protocol versions.

OMA DM Version 2.0 simplifies the transaction model of the DM Command and package flows.

OMA DM Version 2.0 also introduces web-based user ineraction using the SHOW command, and allows management data delivery using HTTP.

5. DM 2.0 Protocol

OMA DM 2.0 is the next generation of the OMA DM 1.x Protocol, and provides the interface between the DM Server and the DM Client to manage the device. OMA DM 2.0 leverages a RESTful architecture for scalability and management performance, and is also designed to work efficiently on less capable devices. The following sub-sections present a high-level overview of the OMA DM 2.0 Protocol.

5.1 Protocol Overview

5.1.1 Transaction Model

OMA DM 2.0 Protocol runs within the context of a DM Session. DM Sessions are always initiated by the DM Client. However, a DM Server can trigger the DM Client to initiate a DM Session by sending the DM Notification to the DM Client. Once a DM Session is established, the DM Server sends DM Commands to the DM Client and receives responses from the DM Client. The DM Client also informs the DM Sever about events that have occurred on the device, via Generic Alerts. Only the DM Server sends DM Commands to the DM Client, and the DM Client cannot send any DM Commands. The DM Server terminates the DM Session by sending the END command to the DM Client.

The OMA DM 2.0 supports the notion of DM Packages. The originator of this DM package should wait for the response from the recipient before sending another DM package. Since processing of DM packages can consume unpredictable amount of time, the OMA DM Protocol does not specify any timeouts between DM packages.

The DM package can only be transferred on top of the HTTP compatible protocols.

5.1.2 Security Considerations

Management of device is a sensitive operation which can involve secrets and confidential data (i.e. password), so it is recommended to perform DM operation in a secure and authenticated context. OMA DM specifications do not provide the full security features for the secure management operations, provided that underlying layer mechanisms can be employed. Plese refer to section 9 for Security.

5.1.3 DM Protocol Interface

Please refer to the OMA DM 2.0 AD [DM-AD] specification for protocol interfaces details.

5.1.4 Management Data Delivery using HTTP

The DM Server can send specific DM Commands (defined in this specification) to the DM Client for retrieving or sending management data from or to the Data Repository: this management is carried over HTTP protocol.

In this specification some serialization formats (MIME) allowing the transfer of management data over HTTP protocol are defined, but others are not precluded.

5.1.5 Web-based User Interaction

OMA DM 2.0 supports the web-based user interaction, which enables the DM Server to use the web pages to interact with the user. For this reason, the Web Browser Component and the Web Server Component are introduced, and a DM Command SHOW is specified. The interface between the Web Browser Component and the Web Server Component is out-of-scope of this specification, and the UI session for performing the user interaction is separated from the DM session. The Web Browser Component can be integrated in the DM Client or can run as a standalone application. More details are specified in the section 5.3.7.

5.2 Package Flow

The package flow between the DM Server and the DM Client are shown below. The contents for each DM Package are described in the section 8.

The DM Server and the DM Client MUST support this package flow.

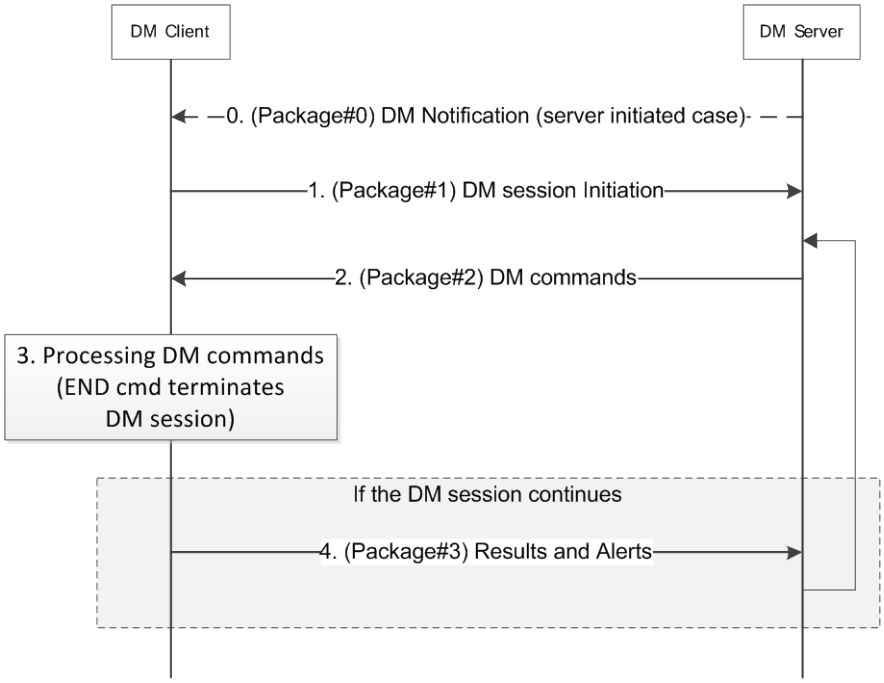


Figure 1: DM Package Flow

Step 0 (Package#0): the DM Server requests the DM Client to start a DM Session by sending the DM Notification. This DM Notification is called "Package#0". This is an optional package flow since the DM Notification is not needed in the client initiated case.

Step 1 (Package#1): the DM Client initiates the DM session sending the "Package#1": the package can contain the information of the MOs supported by the DM Client, that can be used by the DM Server for management operations.

Step 2 (Package#2): after receiving the Package#1 or the Package#3, the DM Server sends the management commands to the DM Client: the package containing the DM Commands is called "Package#2".

Step 3 (Command Processing): the DM Client processes sequentially the DM Commands received in the Package#2;it may interact with external components other than the DM Server (e.g., the Web Browser Component in case of the SHOW command, the Data Repository in case of the HGET/HPOST/HPUT command).

Step 4 (Package#3): if the Package#2 does not include the END command, the DM Session continues and the DM Client responses to the DM Server sending the "Package#3", which contains the results of the DM Command execution. After the Package#3, the package flow goes back to the Step 2, with the Package#2 sent by the DM Server.

The DM Session is defined by the Package#1 to Package#3 flow; the Package#0 and the HTTP transactions initiated by DM Client to execute the provided DM Commands are not formally part of the DM Session.

5.2.1 Package#0: DM Notification

Many devices cannot continuously listen for connections from a management server. Other devices simply do not wish to "open a port" (i.e. accept connections) for security reasons. However, most devices can receive unsolicited messages, sometimes called "notifications". Some handsets, for example, can receive SMS messages. Other devices may have the ability to receive other, similar datagram messages. A DM Server can use this notification capability to cause request the DM Client to initiate the DM Session.

DM Notification consists of a number of mandatory parameters, called Headers, and a number of optional parameters, called Options: the number of Options is determined by the Header. The format of this package is specified in the section 7.1.

The DM Server MUST support the Package#0. If the DM Client support at least one of the transport mechanism described in Appendix E, then it MUST support the Package#0.

5.2.1.1 Package Headers

All headers MUST be present in the DM Notification: the DM Server and the DM Client MUST support the Package headers described in this section.

The following headers are defined in this specification:

- Version (VER)

  The VER field specifies the version of the DM Notification sent by the DM Server. The value for this specification MUST be 0x02. Note that this is not the DM Protocol version, but the DM Notification version.

- Options Count (OC)

  The OC field specifies the number of Options in the DM Notification.

5.2.1.2 Package Options

Each Option MUST be uniquely identified with an Option Number, and MAY be present in the DM Notification. The standard Options defined in this specification are as follows:

Option No	Name	Value Format	DM Client Support	DM Server Support	Occurrence
1	SERVER-ID	String	Mandatory	Mandatory	ZeroOrOne
2	PREFERRED-CON-TYPE	Opaque	Mandatory	Mandatory	ZeroOrOne
3	NOTIFICATION-ID	Uint	Optional	Mandatory	ZeroOrOne
4	SHA256-DIGEST	Opaque	Optional	Mandatory	ZeroOrOne
5	TIMESTAMP	Opaque	Optional	Mandatory	ZeroOrOne
6	REQ-MOS	Null	Mandatory	Mandatory	ZeroOrOne

Table 1: Standard Options

Option carries the value whose format MUST be one of the followings:

- Uint: A non-negative integer which is represented in network byte order using the bytes which Option Length decides. The Option Value range is calculated by 2 to the power of Option Length in bit. For example if the Option Length is 2, Option Value range is 0-65535 in decimal.

- String: A Unicode string which is encoded using UTF-8 [RFC3629] in Net-Unicode form [RFC5198]. Note that ASCII strings (that do not make use of special control characters) are always valid UTF-8 Net-Unicode strings.

- Opaque: An opaque sequence of bytes. This type could be used when the other types than Uint or String is required. How to handle this type depends on the Option using this type.

- Null: The Option carries no value.

The detailed explanations for each Options as follows:

- Server ID Option (SERVER-ID)

  The SERVER-ID Option specifies the Server Identifier of the DM Server. This is the same identifier as in the DM Account MO. This Option MAY NOT be present if the DM Client is able to discover the Server Identifier of the DM Server that sent the DM Notification.

- Preferred Connection Type Option (PREFERRED-CON-TYPE)

  The PREFERRED-CON-TYPE Option specifies the preferred connection that the DM Client is requested to use for connecting to the DM Server. If multiple preferred connections are specified, the connection which appears first is to have higher priority over the rest of available bearers. The DM Client SHOULD use the preferred connections with higher priority first if they are available. If none of indicated preferred connections is available, the DM Client SHOULD wait until one of them becomes available unless "ANY_AVAILABLE" is used. If "ANY_AVAILABLE" is used, it MUST be put at the end of the preferred connection, and the DM Client SHOULD select any connection type that is currently available if all higher prior connections are not currently available.

  The values of this Option MUST be one of the following:

Value	Semantics	Description
0x00	ANY_AVAILABLE	Indicates the preferred connection is anything currently available
0x01	CELLULAR	Indicates the preferred connection is cellular, e.g. GSM/CDMA/UMTS/LTE
0x02	WLAN	Indicates the preferred connection is WLAN. e.g. IEEE 802.11 a/b/g/n/ac
0x03	WIRELINE	Indicates the preferred connection is wireline

- Notification ID Option (NOTIFICATION-ID)

  The NOTIFICATION-ID Option specifies 16-bit unsigned integer used for detecting the duplication of the DM Notification. This Option MAY NOT be presented if the underlying transport provides the functionality to discard the duplicated DM Notification. The length of this Option MUST be 2 bytes.

  The DM Client might receive the same DM Notification multiple times, and the duplication can be detected by this Option and the Server Identifier of the DM Server that sends the DM Notification. The DM Client MUST drop the duplicated DM Notification.

  The DM Server MUST properly set this Option for the DM Client to detect the duplication. For instance, the DM Server may sequentially increase this field for each separate DM Notification.

- SHA256 Digest Option (SHA256-DIGEST)

  The SHA256-DIGEST Option specifies the digest for the DM Notification. The length of this Option MUST be 32 bytes. The DM Sever MUST set this Option as follows:

  – Step1: The DM Server prepares the DM Notification with this Option. The value of this Option MUST be initially set to all zero (zero-digest), and all other Options MUST be properly set.

  – Step2: The DM Server calculates the SHA256 digest according to [RFC6234]. The Input to the hash function MUST be the concatenation of the DM Server secret and the DM Notification (i.e., Digest=Hash(server-secret|notification-message|auth-data). Note that the DM Notification contains all zero for the digest (zero-digest) at this step.

  – Step3: The DM Server replaces the zero-digest with the computed digest.

  If the DM Account MO is used for providing the credentials for this Option, the server-secret MUST be provided at the *<x>/Credentials/Noti/AuthSecret* node in the DM Account MO, and the auth-data MUST be provided at the *<x>/Credentials/Noti/AuthData* node in the DM Account MO.

When receiving the DM Notification with this Option, the DM Client MUST ignore the DM Notification for below cases:

- The DM Server secret is not properly provided at the *AuthNoti* sub-tree in the DM Account MO, or
- The digest in the Option is incorrect.

- Timestamp Option (TIMESTAMP)

  The TIMESTAMP Option specifies the time when the DM Server sends the DM Notification. This time information can be used to prevent the reply attacks. The value of this Option MUST be the time in POSIX format [POSIX].

  When receiving the DM Notification with this Option, the DM Client MAY ignore the DM Notification if the time indicates in this Option is too old (implementation specific decision).

- Request MOS Option (REQ-MOS)

  The REQ-MOS Option requests that the MOS array MUST be sent in the Package #1 as specified in the section 5.2.2. This Option carries no value.

5.2.2 Package#1: DM Session Initiation by DM Client

DM Session is initiated by the DM Client which sends the Package#1 to the DM Server.

The DM Server and the DM Client MUST support this package.

This package MUST be implemented as HTTP POST Request, and the *OMADM-DevID* HTTP header MUST be presented to contain the value of the *DevInfo/DevID* node in the DevInfo MO (section 12.1).

The Package#1 is used by the DM Client:

- To send to the DM Server the list of supported MOs if the `REQ-MOS` Option is used in the Package#0: in this case one "MOS" array MUST be included in the Package#1. Each item of the array, representing a supported MO, MUST contain the following information:
  - The link to the DDF file. Details for DDF can be found at the section 6.4.
  - The MOID of the MO
  - The array containing the list of the MIID for the MOID

  The MOID MUST be provided to the DM Server only if the DM Server provisioned the MOID during the bootstrap. Once the MOID is provided to the DM Server, the MIID for the MOID MUST be provided to the DM Server if the DM Server has any permission for the MO instance.
- To inform the DM Server of any Client Initiated Alerts; one "Alert" array MUST be included:
  - If the DM Session has been started as response to a DM Notification, then the DM Client MUST include a Generic Alert with "urn:oma:at:dm:2.0:ServerInitiatedMgmt" type and the Data property of the Generic Alert MUST include the NOTIFICATION-ID provided by the DM Notification.
  - If the DM Session has been started by the DM Client independently, then the DM Client MUST include a Generic Alert with "urn:oma:at:dm:2.0:ClientInitiatedMgmt" type.
  - If the DM Session has been started as a consequence of successful bootstrap, then the DM Client MUST include a Generic Alert with "urn:oma:at:dm:2.0:BootstrapComplete" type.

5.2.3 Package#2: DM Commands from DM Server to DM Client

The DM Server sends the Package#2 to the DM Client as a response to the Package#1 or the Package#3, in order to send the DM Commands. Multiple DM Command can be listed, and the DM Commands MUST be ordered in a sequence since the

DM Client MUST sequentially process the DM Command according to this order. The same DM Command can be listed multiple times also.

The DM Server and the DM Client MUST support this package.

This package MUST be implemented as HTTP Response [RFC2616] to HTTP Request that carries the Package#1.

The Package#2 is used by the DM Server to send to the DM Client the ordered list of DM Commands to be executed; each item of the list contains:

- The DM Command (see Chapter 5.3)
- The list of the parameters for the DM Command

5.2.4 Package#3: Response Package from DM Client to DM Server

The DM Client sends the Package#3 to the DM Server as a response to the Package#2. If the Package#2 includes the END command, this Package#3 MUST NOT be sent.

The DM Server and the DM Client MUST support this package.

This package MUST be implemented as HTTP POST Request, and the *OMADM-DevID* HTTP header MUST be presented to contain the value of the *DevInfo/DevID* node in the DevInfo MO (section 12.1).

The Package#3 is used by the DM Client:

- To send to the DM Server the list of status codes for the DM Commands indicated in the Package#2; each item of the list MUST contain the status code: see B.2.1 for the valid list of status codes. Additional information (e.g., the stored location of data for the HGET command) MAY be returned with the status code.
- To send to the DM Server new optional Client Initiated Alerts raised during the session: in this case one “Alert” array MUST be included.

5.3 DM Commands

This specification supports the following DM Commands.

Command	Description	DM Server support	DM Client support
HGET	The DM Server uses this command to requests the DM Client to retrieve data from the Data Repository using HTTP GET, and add or replace the received data into the DM Tree	MUST	MUST
HPUT	The DM Server uses this command to request the DM Client to send data to the Data Repository using HTTP PUT	MUST	MUST
HPOST	The DM Server uses this command to request the DM Client to send data to the Data Repository using HTTP POST	MUST	MUST
DELETE	The DM Server uses this command to delete data in the DM Tree	MUST	MUST
EXEC	The DM Server uses this command to execute an executable node in the DM Tree	MUST	MUST
GET	The DM Server uses this command to retrieve data from the DM Tree. The DM Client sends the data within the current DM Session	MUST	SHOULD
SHOW	The DM Server uses this command to initiate a UI Session between the Web Browser Component and the Web Server Component	MUST	SHOULD
CONT	The DM Server uses this command for the DM Client to continue the DM Session with the specified DM Server URI	MUST	MUST

END	This command is used by the DM Server to terminate the DM session	MUST	MUST
DEFAULT	Configure the DM Client to use a specific address to capture configuration if that is missing in the device for an specific MOID	MUST	SHOULD
SUB	The DM Server uses this command to request to the DM Client to report (subscribe) changes in the DM Tree part identified by the provided ClientURI	SHOULD	SHOULD
UNSUB	The DM Server uses this command to request to the DM Client to revoke previous subscription to notification for changes in the DM Tree part identified by the provided ClientURI	SHOULD	SHOULD

5.3.1 HGET

This command supports the following parameters:

Parameter	Format	Occurrence
ServerURI	ServerURI as specified in the section 6.2	One
ClientURI	ClientURI as specified in the section 6.1	ZeroOrOne

The DM Client will perform a *HTTP GET* request against the provided ServerURI to retrieve MO data, and then will add or replace the received data into the DM Tree. The ServerURI refers to a location in the Data Repository. If the ClientURI is specified, the DM Client will try to store the retrieved data at the ClientURI position replacing all existing data, if any.

If the ClientURI is not specified, then the DM Client will choose the location where to store the data and this location MUST be returned with the status codes. Multiple ClientURIs MUST be returned if retrieved data is stored at the multiple locations in the DM Tree.

This command can be used by the DM Server to create an MO instance in the device. When an MO instance is created, the MIID MUST be assigned by the DM Client.

The message flows and the detailed explanations for this DM Command are shown below. Please note that below message flow is simplified not showing the interaction between the DM Client and the DM Server.

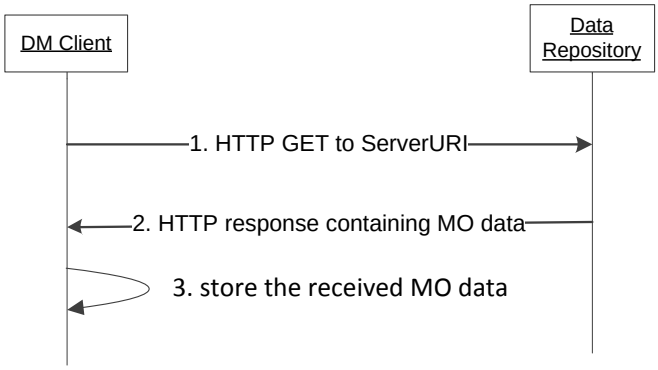


Figure 2: Example for the HGET command

Before the Step 1**,** the DM Server sends HGET command to the DM Client using the Package#2.

Step 1: The DM Client sends the HTTP request (HTTP GET) to the ServerURI. In the HTTP request message, *Accept* HTTP header can indicate the Media Types that the DM Client supports.

Step 2: The Data Repository returns the HTTP response containing the MO data requested. The Data Repository serves the DM Client with the proper MO data format, indicated by *Content-Type* HTTP header, based on the *Accept* HTTP header specified in the previous Step 1.

Step 3: The DM Client stores the received MO data in the DM Tree.

5.3.2 HPUT

This command supports the following parameters:

Parameter	Format	Occurrence
ServerURI	ServerURI as specified in the section 6.2	One
ClientURI	ClientURI as specified in the section 6.1	OneOrMore

The DM Client will perform a *HTTP PUT* request against the provided ServerURI (identifying a location in the Data Repository) sending the requested data whose location is identified by all ClientURIs. The data requested MAY be formatted according to the section 7, but other formats are not precluded.

The message flows and the detailed explanations for this DM Command are shown below. Please note that below message flow is simplified not showing the interaction between the DM Client and the DM Server.

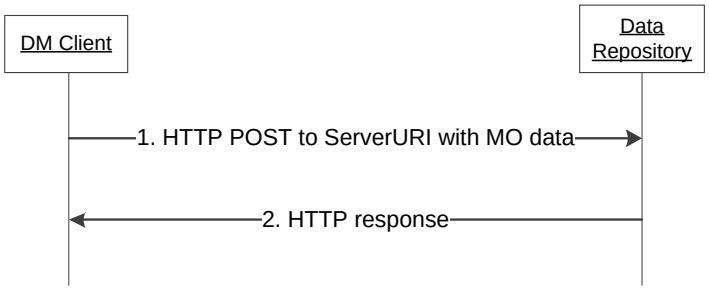


Figure 3: Example for the HPUT command

Before the Step 1, the DM Server sends the HPUT command to the DM Client using the Package#2.

Step 1: The DM Client performes the *HTTP PUT* request against the ServerURI.

Step 2: The Data Repository returns the HTTP response.

5.3.3 HPOST

The flow for this DM Command is the same of HPUT DM Command, with the difference that the DM Client sends the data performing a *HTTP POST* request.

5.3.4 DELETE

This command supports the following parameters:

Parameter	Format	Occurrence
ClientURI	ClientURI as specified in the section 6.1	One

The DM Client will delete the node identified by the ClientURI in the DM Tree. All child nodes will be deleted as well if the node identified by the ClientURI is an interior node with child nodes.

5.3.5 EXEC

This command supports the following parameters:

Parameter	Format	Occurrence
ClientURI	ClientURI as specified in the section 6.1	One
ServerURI	ServerURI as specified in the section 6.2	ZeroOrOne

The DM Client will perform the EXEC operation on the node identified by the ClientURI. If the ServerURI is specified then the asynchronous reporting mechanism MUST be used and the Generic Alert MUST be sent to the ServerURI in a new DM Session (i.e., in the Package#1 or in the Package#3).

5.3.6 GET

This command supports the following parameter:

Parameter	Format	Occurrence
ClientURI	ClientURI as specified in the section 6.1	One

The DM Client will send to the DM Server the requested data identified by the ClientURI. Please refer to Appendix F for the execution of this command.

5.3.7 SHOW

This command supports the following parameter:

Parameter	Format	Occurrence
ServerURI	ServerURI as specified in the section 6.2	One

SHOW command can be used for the DM Server to communicate with the user via the web-based user interaction. This user interaction is accomplished via a UI session, separated from the DM Session, between the Web Browser Component and the Web Server Component. The DM Server and the DM Client do not have knowledge of the context of the UI session, and the results of the UI session is transmitted using the internal interface between the Web Server Component and the DM Server. The DM Server can decide the next management operations according to the results of the user interaction. The internal interface between the Web Server Component and the DM Server is out of scope and left to implementations.

The ServerURI MAY contain the necessary information to identify each UI session (i.e. URI query string), and this information can be used when the DM Server retrieves the results for the user interaction from the Web Server Component.

Once received the SHOW command, the DM Client SHALL initiate the Web Browser Component to load the ServerURI and show the web pages to the user. After the Web Browser Component is successfully initiated, the DM Client MUST process the next DM Command not waiting for the conclusion of UI session: In fact, the DM Client cannot know when the UI session has been finished since the user interaction might consist of several web pages. If the DM Client fails to initiate the Web Browser Component, the DM Client MUST process the next DM Command.

The message flows for this DM Command are shown below with detailed explanations.

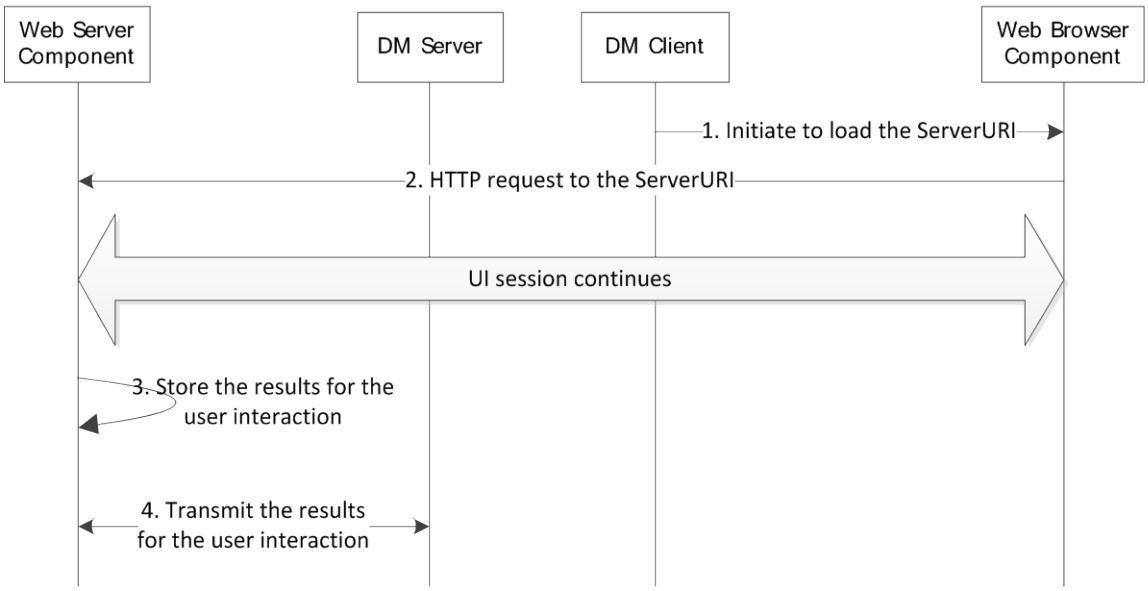


Figure 4: Example for the SHOW command

Before the Step 1, the DM Server sends the SHOW command to the DM Client using the Package#2. The SHOW command takes the ServerURI parameter.

Step 1: to process the SHOW command, the DM Client initiates the Web Browser Component to load the ServerURI and show the web pages to the user. The ServerURI can contain the information to identify this UI session, for example, the query component "?uiid=1234" (user interaction ID). The DM Client continues to the next DM command after the Web Browser Component is initiated.

Step 2: the Web Browser Component sends the HTTP request to the ServerURI, and the user interaction begins; it might consists of several web pages. This user interaction can takes time, or user might not respond.

Step 3: once the user interaction is finished, the Web Server Component stores the results of the user interaction. In case that the user interaction is failed, the error code can be also stored.

Step 4: the results of the user interaction are transmitted between the Web Server Component and the DM Server via an out-of-scope interface.

5.3.8 CONT

This command supports the following parameters:

Parameter	Format	Occurrence
ServerURI	ServerURI as specified in the section 6.2	One

This command is used by the DM Server to inform the DM Client to continue the current DM Session against the specified ServerURI. Once received the CONT command, the DM Client MUST send any response packages (i.e., the Package#3) to the specified ServerURI, and MUST keep using this ServerURI for all further response packages in the same DM Session.

5.3.9 END

This command supports no parameters.

The DM Server MUST send this command to terminate the current DM session. On receiving this command, the DM Client MUST process all commands included in the DM package, but MUST NOT return any status and results back to the DM Server.

5.3.10 DEFAULT

This command supports the following parameters:

Parameter	Format	Occurrence
ServerURI	ServerURI as specified in the section 6.2	One
MOID	MOID	One

The DM Client stores the ServerURI and can use it when needs configuration for a specific MOID, for example when an application running locally on the Device requests information about a specific MOID which is not available in the DM Tree. In this case, the DM Client MUST perform the operations specified for HGET DM Command against the stored ServerURI, providing no status back to the DM Server.

When the DM Client receives a DEFAULT command for an already provisioned MOID, the old ServerURI MUST be replaced with the new ServerURI: that means that the DM Client is able to store just one ServerURI per each supported MOID.

5.3.11 SUB

This command supports the following parameters:

Parameter	Format	Occurrence
ServerURI	ServerURI as specified in the section 6.2	One
ClientURI	ClientURI as specified in the section 6.1	One

The DM Server will subscribe to any change occurring in any part of the tree identified by the ClientURI. When a change occurs, the DM Client will perform a *HTTP PUT* request against the provided ServerURI sending the subtree indentified by the ClientURI. The data MAY be formatted according to the section 7, but other formats are not precluded: if the format is not supported by DM Server, the DM Server MUST return “415 Unsupported Media Type” response code and MUST include the *OMADM-Accept* HTTP header (see section 5.5) indicating the supported MIMEs, then the DM Client SHOULD retry the request using one of the indicated MIMEs, if supported.

5.3.12 UNSUB

This command supports the following parameter:

Parameter	Format	Occurrence
ClientURI	ClientURI as specified in the section 6.1	One

The DM Server will unsubscribe to a previous SUB command identified by the ClientURI: the ClientURI MUST be the same one used in a previous SUB command, otherwise the DM Client MUST return “404 Not Found” response code.

If the DM client support the SUB command, then it MUST support the UNSUB command.

5.4 Generic Alert

The DM Server and the DM Client MUST support the Generic Alert mechanism.

The protocol defines a Generic Alert message for alerts generated by the DM Client that can have a relation to a MO: in this case the SourceURI property MUST identify the address to the MO.

Anytime after the Generic Alert is generated, the DM Client MAY send a Generic Alert message to the DM Server using the Package#1 or the Package#3. The Generic Alert message SHALL only be sent from the DM Client to the DM Server.

Generic Alerts MAY have additional requirements for the format and content of the Data property: in case a unrecognized AlertType or a unrecognized Data property is received, the DM Server MUST silently ignore the Generic Alert. The DM Server MUST NOT send any status codes for the Generic Alerts back to DM Client, regardless of that the DM Server successfully processes the Generic Alert or not.

The following table summarizes the properties supported by a Generic Alert:

Property	Description	Occurrence
AlertType	The type of the Generic Alert	One
SourceURI	The address to the node in the MO that is related to this Generic Alert	ZeroOrOne
TargetURI	The additional address related to the Generic Alert. This MUST be a ClientURI as specified in the section 6.1. The usage of the TargetURI is not specified in this specification.	ZeroOrOne
Mark	The importance level. The following values are defined: "fatal", "critical", "minor", "warning", "informational", "harmless" and "indeterminate". If the parameter is omitted then the default importance level "informational" is assumed.	ZeroOrOne
DataType	The Media Type of the Data content. This property MUST be present if the Data property exists.	ZeroOrOne
Data	The additional data for the Generic Alert. The format and the content of the Data are not specified in this specification	ZeroOrOne

OMA DM 1.x Protocol also specifies the Generic Alert that is widely used in MOs. The interworking issues between the OMA DM 2.0 Generic Alert and the OMA DM 1.x Generic Alerts are discussed in the section 13.1.

5.4.1 Asynchronous Reporting for DM Commands

The DM Server and the DM Client MUST support Synchronous and Asynchronous reporting mechanisms.

Synchronous reporting MUST be used for all DM commands except the EXEC command. For the EXEC command, the DM Client returns the status of the EXEC command either synchronously (i.e. the final status code in the response package) or asynchronously, via the Generic Alert mechanism. If the ServerURI is present in the EXEC command, the DM Client MUST use the asynchronous reporting; otherwise it is up to the DM Client to decide which reporting mechanism is used for the EXEC command depending on the nature of the management operation.

If the asynchronous reporting is used, the DM Client MUST return the status code "202 Accepted" response code in the response Package#3. After completing the EXEC command, the final status code will be returned in the Generic Alert either in the Package#1 or Package#3. Additional Generic Alert data for the asynchronous reporting might be defined for each EXEC command, which is out-of-scope of this specification.

5.5 Object serialization format handshake

According to [RFC2616], the DM Client SHOULD use the *Accept* HTTP header in order to inform the DM Server in a HTTP request about the supported MIMEs to be used to format the content of the HTTP response. In case the DM Server does not support any of the MIMEs requested by the DM Client, “406 Not Acceptable” response code MUST be returned in the HTTP response; in case the format used by DM Client in a HTTP request if not supported by DM Server, “415 Unsupported Media Type” response code MUST be returned in the HTTP response.

If the DM Server supports other MIME than the ones mandated by this specification (see Section 7), it MUST use the *OMADM-Accept* HTTP header in order to inform the DM Client in a HTTP response about the supported MIMEs to be used to format the content of following HTTP requests (HTTP PUT and POST): for example, the DM Server can include it the HTTP request carrying the Package#2 in order to inform the DM Client about the supported MIMEs which can be acceptable for formatting data requested by a HPUT or HPOST DM Command. The *OMADM-Accept* HTTP header format is the same of HTTP *Accept* header [RFC2616]. If the DM Client doesn't support any of the MIME requested by DM Server, the "406 Not Acceptable" response code (see Appendix C) MUST be returned in the Package#3 for that DM Command.

5.6 HTTP Error handling

If the DM Client receives a "4XX" response code to a HTTP request and it is not able to recover from it, the DM Session MUST be considered as aborted; the DM Client MUST NOT retry to connect to the DM Server.

If the DM Client receives a "5XX" response code to a HTTP request or if the HTTP request expires (due to connectivity issues), the DM Session MUST be considered as aborted; the DM Client MUST NOT retry to connect to the DM Server.

6. Device Management Object

6.1 ClientURI

OMA DM 2.0 supports the ClientURI addressing that can be used to identify each node(s) in the device. The addressing is represented as a URI format that is compatible with [RFC3986]. The DM Server MUST support the ClientURI addressing scheme and all the components described. The DM Client MUST support the ClientURI addressing scheme but SHOULD support the query component, the x-name component and the wildcard. The DM Server MUST generate the ClientURI as specified in this section to manipulate the DM Tree.

6.1.1 ClientURI Addressing Scheme

ClientURI MUST follow the format represented in ABNF [RFC4234]: note that `ALPHA` and `unreserved` are adapted from [RFC3986].

```
ClientURI          = MOID "/" mo-inst path-from-miroot [ "?" query ]
mo-inst            = MIID / x-name / "" / "*"
path-from-moroot   = "/" / 1*("/" node-name)
node-name          = real-name / x-name / "*"
real-name          = 1*unreserved
x-name             = "(" ALPHA ")"
```

The MOID and mo-inst (if present) components point to an MO instance: the path-from-moroot component points to node(s) in the MO instance by describing the relative path from the MO instance root node.

6.1.1.1 The MOID and mo-inst Component

The MOID component represents the MOID of the MO instance whose node(s) the ClientURI targets to. The mo-inst component can be one of the followings:

- MIID: the MIID of the MO instance whose node(s) the ClientURI targets to. The MIID is unique within MO instances for the same MOID in the DM Tree: the MOID and MIID pair identified uniquely an MO instance in the DM Tree
- x-name: this is represented by three characters such as "(x)" or "(y)", and the ALPHA component in the x-name component is uniquely assigned within all x-name component in one ClientURI. The corresponding nv fields are provided to resolve the actual MO instance
- empty string (i.e., ""): this can be used when there is only one MO instance for the specified MOID
- wildcard ("*"): all MO instances with the specified MOID are addressed

6.1.1.2 The path-from-moroot Component

The path-from-moroot component can be the single character "/" addressing the MO instance root node, or can be organized as the sequence of node-name starting from the child node of the MO instance root node.

The node-name can be one of the followings:

- real-name: the actual node name
- x-name: this is represented by three characters such as "(x)" or "(y)", and the ALPHA component in the x-name component is uniquely assigned within all x-name components in one ClientURI. The corresponding nv fields MUST are provided to resolve the actual node
- Wildcard ("*"): all nodes at the specified position are addressed

Named nodes are addressed by the real-name component; while un-named nodes can be addressed by any (i.e., the real-name component, the x-name component and the wildcard). Note that the ClientURI uses ' (' and ')' instead of '<' and '>' because the usage of '<' and '>' characters is not compatible with [RFC3986].

6.1.1.3 The query Component

The query component contains additional information for the ClientURI: it is indicated by the first question mark and terminates with the end of the ClientURI. The query component is organized as the sequence of "field=value" separated by the ampersand "&".

The following table defines the standard fields that can be used in the query component, but vendor-specific extensions for additional fields are also possible. The DM Client MUST ignore unrecognized fields.

Fields	Format	Description	Occurrence	Applied DM Cmds
level	lv=<n>	<n> represents a positive integer. If this field is specified, the DM Client MUST only include n sublevels of child nodes; otherwise, all child nodes MUST be included.	ZeroOrOne	GET, HPOST, HPUT
node-value	nv=<path>:<val>	This field can be used to resolve the x-name component. <path> MUST begin with "(" ALPHA ")" where the ALPHA component indicates the corresponding x-name component in the ClientURI. Starting from the corresponding x-name component, <path> indicates the relative path to the leaf node. Multiple nv fields can be specified for the same ALPHA component. <val> MUST specify the value for the leaf node specified by the <path>. The nv field can be said to be "satisfied" if the node specified by the <path> has the value specified by the <val>.	ZeroOrMore	All commands supporting the ClientURI parameter
cache validator	cv=<val>	<val> indicates the cache validator for the node represented by the ClientURI. See the section 12 for details.	ZeorOrOne	GET, HPUT, HPOST

6.1.2 Resolving ClientURI

The DM Client MUST resolve the ClientURI into a unique location if the wildcard is not used in the ClientURI. If the wildcard is used, the ClientURI might be interpreted to identify multiple nodes in the device, and the DM Client MUST return a list of MO data where each member represents each MO data matched by the ClientURI.

This resolving procedure consists of two steps; finding the MO instance(s), and finding the node(s) within the MO instances. The DM Client MUST apply the following procedure to resolve the ClientURI:

Step 1: Finding the MO instance(s) according to the mo-inst component.

The MIID, x-name, empty string and wildcard for the mo-inst component are interpreted as follows:

- MIID: to proceed to the Step2, the DM Client MUST find only one MO instance for the specified MOID and MIID. If no or multiple MO instances are found, an error code MUST be returned
- x-name: to proceed to the Step2, the DM Client MUST find only one MO instance that satisfies all corresponding nv fields for this x-name component. If no or multiple MO instances are found, an error code MUST be returned
- Empty string: to proceed to the Step2, the DM Client MUST find only one MO instance for the MOID. If no or multiple MO instances are found, an error code MUST be returned
- Wildcard: to proceed to the Step2, the DM Client MUST consider all MO instances for the specified MOID. If there is no MO instance for the MOID, an error code MUST be returned

Step 2: Within MO instance(s) found in the Step 1, the DM Client resolves the path-from-moroot component interpretating the real-name, x-name and the wildcard in the path-from-moroot component as follows:

- real-name: the DM Client MUST match the actual node name
- x-name: the DM Client MUST resolve only one node that satisfies all corresponding nv fields for this x-name component; if multiple nodes are resolved, an error code MUST be returned
- Wildcard: the DM Client MUST address all nodes at the specified location

6.1.3 Addressing Examples

In this section, illustrative examples for addressing schemes are shown.

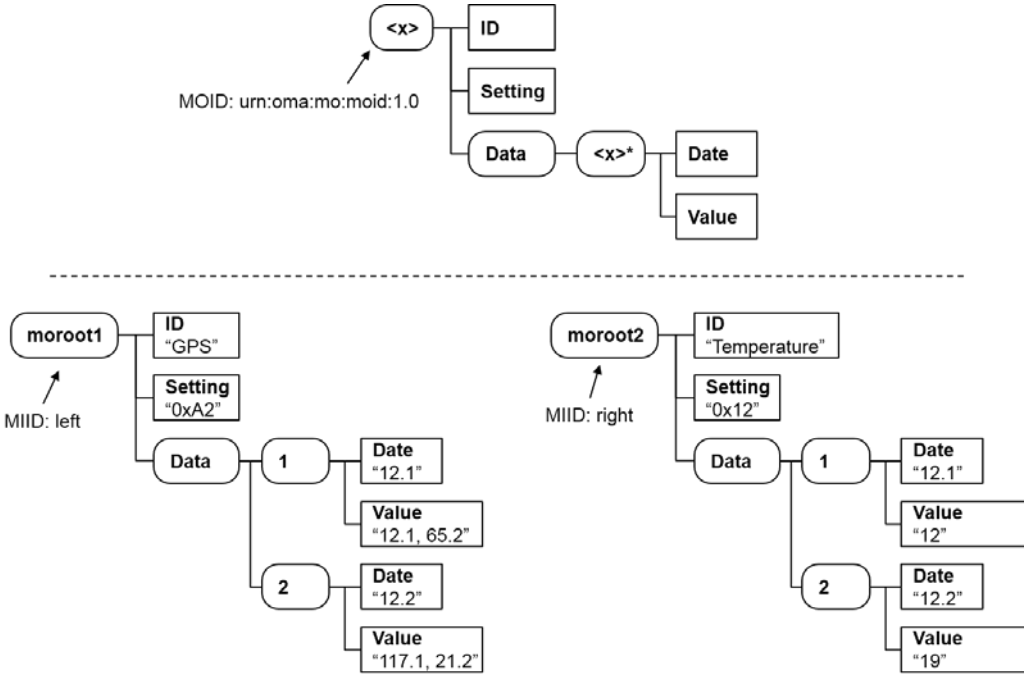


Figure 5: Addressing Scheme Examples; MO definition and two instances for it

- "urn:oma:mo:moid:1.0/left/" identifies the *moroot1* node
- "urn:oma:mo:moid:1.0//" cannot be resolved since there are two MO instances
- "urn:oma:mo:moid:1.0/(x)/?nv=(x)/ID:Temperature" identifies the *moroot2/ID* node
- "urn:oma:mo:moid:1.0/(x)/Data/1/Value?nv=(x)/ID:GPS" identifies the *moroot1/Data/1/Value* node
- "urn:oma:mo:moid:1.0/right/Data/(x)/Value?nv=(x)/Date:12.1" identifies the *moroot2/Data/1/Value* node
- "urn:oma:mo:moid:1.0/(x)/Data/(y)/Value?nv=(x)/ID:GPS&nv=(y)/Date:12.2" identifies the *moroot1/Data/2/Value* node
- "urn:oma:mo:moid:1.0/*/Setting" identifies two nodes; the *moroot1/Setting* and *moroot2/Setting* node
- "urn:oma:mo:moid:1.0/(x)/Data/*/Value?nv=(x)/ID:GPS" identifies two nodes; the *moroot1/Data/1/Value* and *moroot1/Data/2/Value* node

- "urn:oma:mo:moid:1.0/(x)/Data/*/Value?nv=(x)/ID:GPS&nv=(x)/Setting:0xB2" cannot be resolved since there is no MO instances satisfying the two nv fields
- "urn:oma:mo:moid:1.0/*/Data/*/Value" identifies four nodes; the moroot1/Data/1/Value, moroot1/Data/2/Value, moroot2/Data/1/Value, and moroot2/Data/2/Value node

6.2 ServerURI

The ServerURI is a URI compliant to [RFC3986]. The DM Server and the DM Client MUST support the ServerURI.

ServerURI MAY include some parameters within the ' [' character and the ']' character that the DM Client MUST replace with its own values before it is used. After these parameters are replaced, the ServerURI MUST be encoded as a URI as defined in [RFC3986]. All non-supported values MUST be replaced with the integer value '-1'.

The DM Server MUST support the values defined in the following table; the DM Client support is defined in the related column.

Name	Description	DM Client Support
DevID	Unique identifier of the device. If correspondes to the value of the *<x>/DevID* node in the DevInfo MO	MUST
MNC	Home network's Mobile Network Code **[3GPP23.003]** If the DM Client is not associated with a mobile network then this value MUST be -1	MUST
MCC	Home Network's Mobile Country Code **[3GPP23.003]** If the DM Client is not associated with a mobile network then this value MUST be -1	MUST
CMNC	Current Mobile Network Code **[3GPP23.003]** If the DM Client is not associated with a mobile network then this value MUST be -1	MUST
CMCC	Current Mobile Country Code **[3GPP23.003]** If the DM Client is not associated with a mobile network then this value MUST be -1	MUST
SPName	Service Provider Name **[3GPP31.102]** If the DM Client is not associated with a mobile network then this value MUST be -1	MUST

6.3 Management Object Definition

A Management Object is a set of nodes which implements a specific function; the type of the Management Object is defined by the MOID.

The DM Server and the DM Client MUST support the MO definition provided in this section.

The DM Server and the DM Client MAY support any MO (proprietary or stardardized elsewhere) specified using the definition provided in this section.

OMA DM 2.0 does not require the MOs in the DM Client to be organized as a hierarchical tree structure since the nodes are addressed based on the MO instance.

6.3.1 Nodes

Nodes are the entities that can be manipulated by management actions carried over the OMA DM Protocol, and there are two types of nodes; the leaf node and the interior node.

The leaf node has a value assigned to the node, and the value might be small as an integer or large and complex like a background picture or firmware image. OMA DM Protocol is agnostic about the node values and the meaning of the node value is specified by the MO description. A leaf node cannot have a child node. An interior node can have child nodes, and cannot have a node value.

A node is defined by the following properties:

Definition	Description
Name	Indicates the node name. The format of the node name MUST follow the below ABNF format where the unreserved is adapted from **[RFC3986]**: `node_name = '<' 1*unreserved '>' / 1*unreserved` The node name that starts with '<' and ends with '>' represents the un-named node.
Status	It indicates the support requirement. If the value is "Required" then the DM Client MUST support the node (if the parent node is supported); if the value is "Optional", then the node is not unconditional mandatory to support in the implementation.
Occurrence	It indicates the number of instances that MAY occur for the node. The valid values are the followings: • One: The number of node MUST be exact one • ZeroOrOne: the number of nodes is zero or one. The DM Server can add the node when the current occurrence was zero • ZeroOrMore: the number of nodes is zero or more. The DM Server can add the node up to the number of device allows • OneOrMore: the number of nodes is one or more. The DM Server can add the node up to the number of device allows • ZeroOrN: the number of nodes is zero or multiple number. The DM Server can add the node up to the number specified as the part of MO specification • OneOrN: the number of node is one or multiple number. The DM Server can add the node up to the number specified as the part of MO specification
Format	It indicates the format of the data w stored on node. Allowed values are "node", "null", "b64", "bin", "bool", "chr", "int", "xml", "date", "time", and "float". Node with the format "node" is the interior node; otherwise it is the leaf node. The format "null" means the node MUST NOT have the node value.
Min. Access Type	It indicates the operations that the DM Server is allowed to perform on the node. The valid values are the combination of the followings: • Add: the node can be added by the DM Server • Delete: the node can be deleted by the DM Server • Exec: the node can be executed by the DM Server • Get: the node can be read by the DM Server • Replace: the node can be written by the DM Server For example, "Get" means this is read-only node. "Get, Exec" means this is readable and executable, but cannot be written. "Replace" means this is write-only node. If nothing specified, it means that this node is not exposed to the DM Server
Description	It indicates the description for the node

6.3.1.1 Node Access Control

In the following table the relationship between DM 2.0 Command and DM 1.3 Min. Access Type is indicated; in addition, the DM Command are grouped in logical operations.

Logical Operation	DM 2.0 Command	DM 1.3 DDF Min. Access Type
Read	GET/HPUT/HPOST	GET
Write	HGET	ADD
		REPLACE
	DELETE	DELETE
Execute	EXEC	EXEC
Delegate	HGET(*)	

(*): HGET to modify the access rights of the sub-tree addressed by the <MOID>/<MIID>/<x>/Path node

For instance, according to this table, the DM Server is granted to perform a DELETE command involving a node whose Min. Access Type in DDF contains DELETE; the DM Client allows to perform a HGET command to create a new node if Add Access Type is granted in the DDF.

The DM Client MUST allow the execution of a DM Command only if the DM Server has the necessary access rights to perform the operation on all the nodes involved by the DM Command.

6.3.1.2 Permanent node Permanent and Dynamic Nodes

Nodes in the Management Object can be either permanent or dynamic.

Permanent node cannot be created, deleted or modified at run-time by the DM Server: any attempt by the DM Server to delete a permanent node MUST return status "405 Command Not Allowed". The same status code will also be returned for all attempts to modify the name of a permanent node.

Dynamic nodes can be created and deleted at run-time by the DM Servers. The HGET command is used to create new dynamic nodes: the DELETE command is used to delete the dynamic node and all its properties. If an interior node with child nodes is deleted, all child nodes MUST also be deleted.

The permanent node MAY be the child of either a dynamic or a permanent node. In such cases, the permanent child node is created at the same time its parent node is created.

“Scope” property of DDF (see 6.4) specifies if a node is permanent or dynamic.

6.3.1.3 Un-named and Named Node

Un-named nodes act as placeholder in the MO and are instantiated with information when the nodes are created at run-time. An un-named node is identified by the name which starts with '<' and ends with '>' in the MO definition. For example, a common un-named node is "<x>", while an informative name can be given such as "<MOID>" that helps understanding that the node name comes from the MOID. It is recommended to provide description about the rule to adopt for the name during the creation of the node.

Named nodes have fixed name. The node name does not need to be unique within the MO, but it MUST be unique within the child nodes belonging to the same parent node.

6.3.1.4 Node Definition Example

The following table is generally used to specify a node definition:

Including the node name, the path from the MO root node is specified here. For example, "<x>/bar" means the name of this node is "bar" that is located under the MO root node "<x>".

Status	Occurrence	Format	Min. Access Types
Required	OneOrMore	node	Get

The description for the node can be specified here.

6.3.1.5 Graphical Representation of MO

The MO definition can be graphically represented using the following graphical notation.

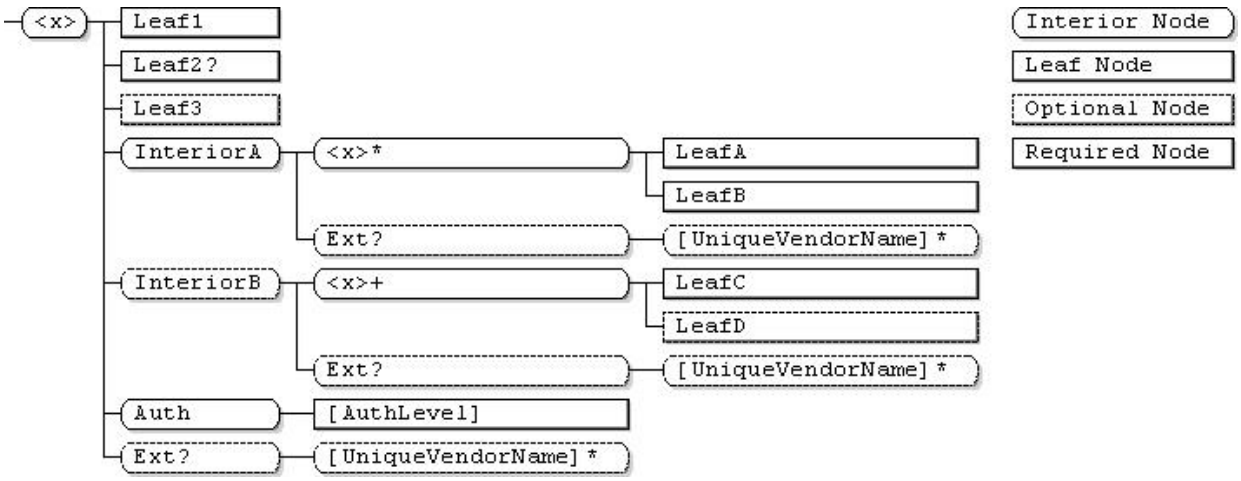


Figure 6: Example of a MO pictured using the graphical notation

Each block corresponds to a defined node, and the text is the name of the node. If the text starts with "<" and ends with ">", it is the un-named node. The nodes which the DM Client is required to support are drawn in the graphical notation with solid line, while nodes whose support is not mandatory for the DM Client are drawn with a dotted line. Leaf nodes are drawn as rectangle while interior nodes are drawn as rectangle with rounded corners.

The occurrence information is appended at the end of node name with the marker. The valid markers and their meaning are defined in the following table. If none of these markers is used the default occurrence is exactly 'One':

Marker	Meaning
+	Occurrence of OneOrMore
*	Occurrence of ZeroOrMore
?	Occurrence of ZeroOrOne
+N	Occurrence of OneOrN
*N	Occurrence of ZeroOrN

Some MOs specify explicit names of nodes but the name is still assigned at run-time. These nodes MAY NOT be described as "<x>" and it is possible to use the syntax [NodeName] where “NodeName” is a logical name for the node. In this case the graphical representation and the DDF File will contain the logical name of the node to improve the readability.

Naturally, this graphical overview does not show all details of the full description, but it provides an overview of the description so that it is easier to find the individual node. Although the figure only provides an overall view of the description, there are still some things worth noticing.

All blocks with names in place occur exactly once, except "Leaf2", "InteriorA/<x>", "InteriorB/<x>", all "Ext" nodes and their children.

"Leaf3", "InteriorB", all "Ext" and their children nodes are optional to be supported by the DM Client.

All nodes whose name starts with "Leaf" and the node "[AAuthLevel]" are leaf nodes. They might contain data but cannot contain child nodes; all other nodes are interior nodes; they cannot contain data but can contain child nodes.

The un-named leaf nodes might be marked with * or +. This means that although the description only contains one node description at this position in the tree, there can be any number of instantiated nodes at run-time, including none in the first case, at least one in the second.

The next figure shows an example of what the above Management Object definition can be instantiated in the device at run-time:

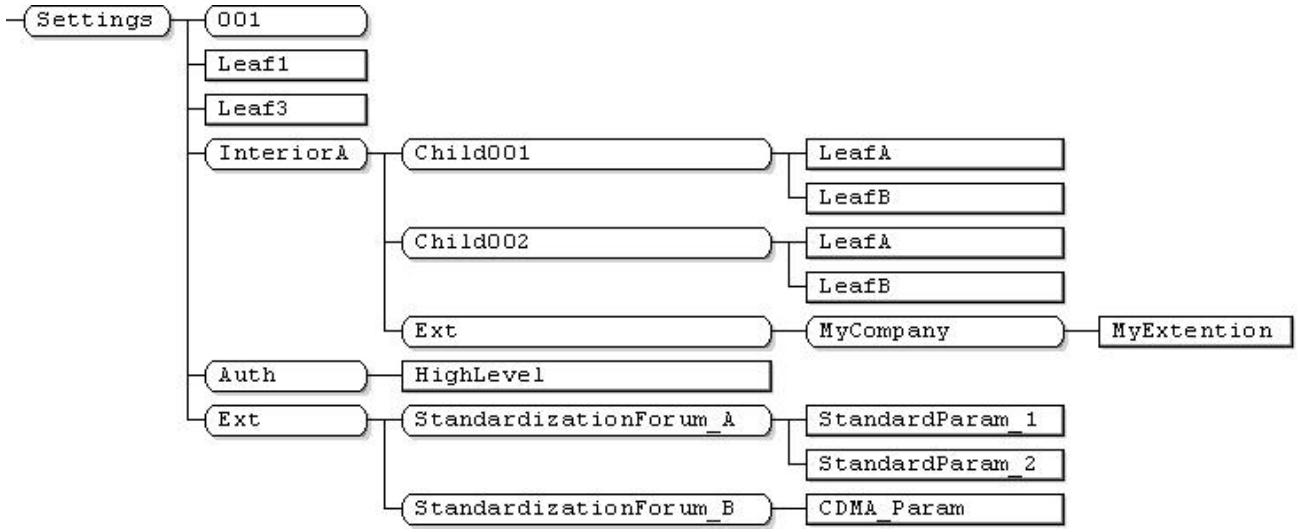


Figure 7: Example of an instance of this MO

The difference between this figure and the previous one is that now the un-named blocks have been instantiated and some optional nodes are not shown.

Note that none of the stored data in the leaf nodes is shown in the figure: only the node names are visible.

6.3.2 Device Description Framework

In an ideal world, all devices would display the same structure and behaviour to the management system. But since different vendors are competing with each other on the market for various kinds of devices, it seems very unlikely that this would ever happen. But management systems still need to understand each individual device even though they do appear to have different internal structures and behaviours.

To address this issue, the concept of a device description framework (DDF) is introduced. In short, this framework prescribes a way for device vendors to describe their devices so that a management system can understand how to manage the device.

The following figure illustrates the conceptual view of how a device description framework is used, however, other approaches for using the DDF is not precluded:

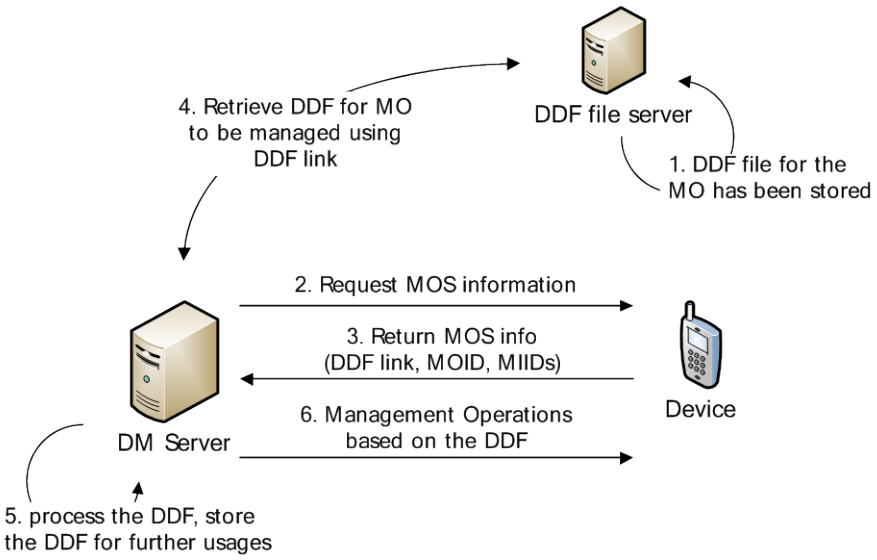


Figure 8: conceptual view of how the device description framework

6.4 Using DM 1.x DDF

OMA DM 2.0 uses the DM 1.3 DDF [DM_1.3], but unnecessary elements in DM 1.3 DDF might be silently ignored. The following table contains the support reference for the DM Server and DM Client.

Elements in DM 1.3 DDF	Descriptions	OMA DM 2.0 Support
MgmtTree	Container for one or more described Management Objects.	Mandatory
VerDTD	Specifies the major and minor version identifier of the OMA DM Device Description Framework specification used to represent the OMA DM description.	Mandatory
Man	Specifies the manufacturer of the device.	Optional
Mod	The model number of the device.	Optional
Node	Specifies a Node.	Mandatory
NodeName	Specifies the name of the described Node.	Mandatory
Path	Specifies the URI up to, but not including, the described Node.	Optional
Value	Specifies a default value for Nodes that are instantiated using the current description.	Optional
RTProperties	Aggregating element for run-time properties, i.e., properties that the Nodes have in a device at run-time. Used to specify which properties the described Node supports at run-time. Can also be used to supply default values for supported run-time properties.	Ignored
ACL	Specifies support for the ACL property. MAY be used to specify a default value for the property.	Ignored
Format	Specifies support for the Format property. MAY be used to specify a default value for the property.	Ignored
Name	Specifies support for the Name property. MAY be used to specify a default value for the property.	Ignored
Size	Specifies support for the Size property.	Ignored
Title	Specifies support for the Title property.	Ignored
TStamp	Specifies support for the TStamp property.	Ignored
Type	Specifies support for the Type property.	Ignored

VerNo	Specifies support for the VerNo property.	Ignored
B64, bin, bool, chr, int, node, null, xml, date, time, float	Specifies the format for the node value.	Mandatory
MIME	Specifies the MIME type of the current Node value.	Optional
DDFName	Specifies the Management Object Identifier of the Management Object rooted at this Node, or is empty.	Mandatory
DFProperties	Aggregating element for device description framework properties, i.e., properties that Nodes have in the device description framework and that are not explicitly present at run-time.	Mandatory
AccessType	Specifies which commands are allowed on the Node.	Mandatory
DefaultValue	The Node value used in a device unless specifically set to a different value.	Optional
Description	The human readable description of the Node.	Optional
DFFormat	The data format of the described Node.	Mandatory
Occurrence	Specifies the number of instances that MAY occur of the Node.	Optional
Scope	Specifies whether this is a Permanent or Dynamic Node.	Optional
DFTitle	The human readable name of the Node.	Optional
DFType	For Leaf Nodes, the MIME type of the Node value. For Interior Nodes, a Management Object Identifier or empty.	Optional
CaseSense	Specifies whether the Node name and names of descendant Nodes in the tree below should be treated as case sensitive or case insensitive.	Optional

7. Object Serialization

The DM Server and the DM Client exchanges the DM Packages as specified in the section 5.4. In this section, the format of the DM Packages is specified.

7.1 Package#0

7.1.1 Binary Format

The DM Server MUST support this format. The DM Client MUST support this format.

The binary format for the DM Notification package consists of a 2 bytes fixed-sized Header, followed by a variable part containing the Options. The byte order for DM Notification package MUST be Big Endian (Network order).

The delivery and transport of the DM Notification is specified in the Appendix D.

7.1.1.1 Package Header

The following figure describes the format of the DM Notification header part:

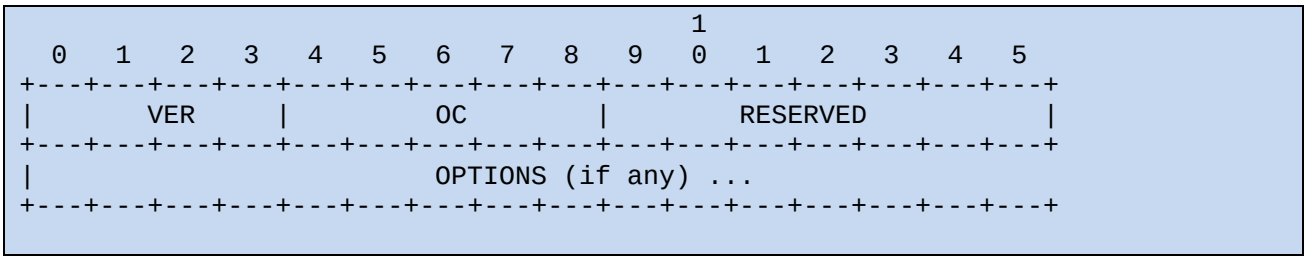


Figure 9: DM Notification package Format

DM Notification header has the fixed size of 2 bytes. The DM Server MUST specify the DM Notification header fields in order as described in the following table:

Header Fields	Bit Length	Descriptions
VER	4	Version of DM Notification package
OC	5	Number of Options included in the DM Notification package
RESERVED	7	Reserved for future Header fields

Table 2: DM Notification Header Fields

7.1.1.2 Package Option

In the DM Notification package, The DM Server MUST specify the Options in the increasing order of their Option Number. Each Option is specified by the Option Delta, the Option Length and the Option Value as shown in the below figure.

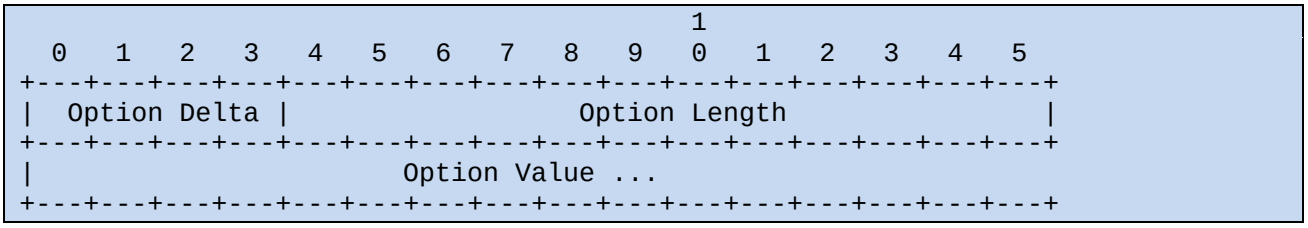


Figure 10: Message Option Format

- Option Delta

The Option-Delta field indicates the difference between the Option Number of this Option and the previous Option. The value of this field is specified using 4 bits. This field is used for calculating the actual Option Number as the sum of its Option Delta and the all preceding Option Deltas. For the first Option in the DM Notification package, the Option Delta becomes the Option Number.

- Option Length

  The Option-Length field specifies the length of the Option Value, in bytes. For instance, if the Option Length value is 3 then the Option Value size is 3 bytes. The value of this field is specified using 12 bits. When the Option carries no value, then DM Server MUST specify the value 0 for the Option-Length field and MUST NOT include Option Value. Hence the shortest Option can be 2 bytes long if it carries no Option Value.

- Option Value

  The format of the Option Value depends on the respective Option.

7.1.2 Other Format

The DM Server and the DM Client MAY implement other serialization format which are out of the scope of this specification.

7.2 Other Packages

7.2.1 JSON Format

The DM Server MUST support this format. The DM Client MUST support this format.

7.2.1.1 Package#1

The Media Type for this format is "application/vnd.oma.dm.initiation+json". The JSON schema [JSON-SCHEMA] for the Package#1 is:

```
{
     "$schema": "http://json-schema.org/draft-04/schema#",
     "title": "OMA DM Package#1 JSON Schema",
     "type": "object",
     "properties": {
          "MOS": {
               "type": "array",
               "items": {
                    "type": "object",
                    "properties": {
                         "DDF": {
                              "type": "string"
                         },
                         "MOID": {"type": "string"},
                         "MIID": {
                              "type": "array",
                              "items": {
                                   "type": "string"
                              }
                         }
                    },
                    "required": ["MIID"]
               }
           },
          "Alert": {
               "$ref": "#alert_json_schema"
          }
     }
}
```

This is an illustrative example of Package#1.

```
POST /dmclient/dm20 HTTP/1.1
Content-Type: application/vnd.oma.dm.initiation+json
Accept: application/vnd.oma.dm.request+json
OMADM-DevID: IMEI:493005100592800
Host: www.devicemgmt.org

{
     "MOS": [
          {
               "DDF": "http://www.vendor.com/DDF/devinfo1.0.ddf",
               "MOID": "urn:oma:mo:oma-dm-devinfo:1.0",
               "MIID": ["miid1"]
          },
          {
               "DDF": "http://www.vendor.com/DDF/oma-sessioninfomo1.0.ddf",
               "MOID": "urn:oma:mo:oma-sessioninfomo:1.0",
               "MIID": ["miid1"]
          }
     ],
     "Alert": [
          {
               "AlertType": "urn:oma:at:scomo:1.0:OperationComplete",
```

```
              "SourceURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Download/Package1/Operations/DownloadInstall",
              "TargetURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Inventory/Deployed/Component1",
              "Mark": "warning",
              "DataType": "text/xml",
              "Data": {
                   "DM1x": "<ResultCode>1200</ResultCode>          <!-- SCOMO
Result Code --><Identifier>Component1ID</Identifier>"
              }
         },
         {
              "AlertType": "urn:oma:at:scomo:1.0:OperationComplete",
              "SourceURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Download/Package1/Operations/DownloadInstall",
              "TargetURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Inventory/Deployed/Component2",
              "Mark": "warning",
              "DataType": "text/xml",
              "Data": {
                   "DM1x": "<ResultCode>1200</ResultCode>          <!-- SCOMO
Result Code --><Identifier>Component1ID</Identifier>"
              }
         }
    ]

}
```

7.2.1.2 Package#2

The Media Type for this format is "application/vnd.oma.dm.request+json". The JSON schema [JSON-SCHEMA] for the Package#2 is:

```
{
     "$schema": "http://json-schema.org/draft-04/schema#",
     "title": "OMA DM Package#2 JSON Schema",
     "type": "object",
     "properties": {
          "CMD": {
               "type": "array",
               "items": {
                    "type": "array",
                    "items": {
                         "description": "DM CMD, parameters in order",
                         "type": "string"
                    }
               }
          }
     },
     "required": ["CMD"]
}
```

This is an illustrative example of package#2.

```
HTTP/1.1 200 OK
Content-Type: application/vnd.oma.dm.request+json

{
     "CMD": [
          ["HPOST",
               "https://DMcontent.DMserver.operator.com/[DevID]",
               "urn:oma:mo:oma-dm-devinfo:1.0//",
               "urn:oma:mo:oma-sessioninfomo:1.0//"],
          ["HGET", "https://DMcontent.DMserver.operator.com/new_mo"],
          ["GET", "urn:oma:mo:oma-sessioninfomo:1.0//CBT"]
     ]
}
```

7.2.1.3 Package#3

The Media Type for this format is "application/vnd.oma.dm.response+json". The JSON schema [JSON-SCHEMA] for the Package#3 is:

```
{
     "$schema": "http://json-schema.org/draft-04/schema#",
     "title": "OMA DM Package#3 JSON Schema",
     "type": "object",
     "properties": {
          "Status": {
               "type": "array",
               "items": {
                    "description": "status codes are ordered in the same
sequence with the DM commands in the Package#2",
                    "type": "object",
                    "properties": {
                         "sc": {
                              "type": "number"
                         },
                         "URI": {
                              "type": "array",
                              "items": {
                                   "type": "string"
                              }
                         }
                    },
                    "required": ["sc"]
               }
          },
          "Alert": {
               "$ref": "#alert_json_schema"
          }
     },
     "required": ["Status"]
}
```

This is an illustrative example of Package#3 that is a response to the example Package#2 in the section 7.2.1.2.

```
POST /dmclient/dm20 HTTP/1.1
Content-Type: application/vnd.oma.dm.response+json
Accept: application/vnd.oma.dm.request+json
OMADM-DevID: IMEI:493005100592800
Host: www.devicemgmt.org

{
     "Status": [
          {"sc": 200},
          {"sc": 200, "URI": ["urn:oma:mo:oma-mo:1.0/miid1/", "urn:oma:mo:oma-
mo:1.0/miid2/"]},
          {"sc": 200}
     ],
     "Alert": [
          {
               "AlertType": "urn:oma:at:dcmo:1.0:OperationComplete",
               "SourceURI": "urn:oma:mo:oma-
dcmo:1.0/Capability123/Operations/Enable",
               "Mark": "warning",
               "DataType": "int",
               "Data": {
                    "DM1x": 1404
               }
          }
     ]
}
```

7.2.1.4 Management Object Serialization

The Media Type for this format is "application/dmmo+json". The JSON schema [JSON-SCHEMA] is:

```
{
     "$schema": "http://json-schema.org/draft-04/schema#",
     "title": "Management Object JSON Schema",
     "type": "array",
     "items": {
          "type":"object",
          "properties": {
               "DDF": {
                    "type": "string",
                    "description": "DDF URL"
               },
               "ClientURI": {
                    "type": "string"
               },
               "CV": {
                    "type": "string"
               },
               "MOData": {
                    "type": "object"
               }
          },
          "required": ["ClientURI","MOData"]
     }
}
```

The following conversion rules are applied:

- **Interior Node**: The interior node is represented as JSON Object containing the NodeName as label and a JSON Object as value. This Object contains the structure of child nodes. If the child nodes are not transferred, the empty JSON object "{}" is specified.
- **Leaf Node**: The leaf node content is represented as JSON object whose type is compliant to the one specified in the related DDF file.

This is the example of JSON serialized MO content:

```
[
     {
          "DDF": "http://www.vendor.com/DDF/devinfo1.0.ddf",
          "ClientURI": "oma:mo:oma-dm-devinfo:1.0//",
          "CV": "686897696a7c876b7e",
          "MOData": {
               "DevInfo": {
                    "DevID": "IMEI:493005100592800",
                    "Man": "Vendor",
                    "Mod": "DM_Client",
                    "DmV": "2.0",
                    "Lang": "en",
                    "DevType": "smartphone",
                    "OEM": "",
                    "FwV": "android4.0.4",
                    "SwV": "Vendor1.2",
                    "HwV": ""
               }
          }
     }
]
```

7.2.1.5 Generic Alert

This is the JSON Schema [JSON-SCHEMA] associated with the Generic Alert:

```
{
     "$schema": "http://json-schema.org/draft-04/schema#",
     "title": "OMA DM Generic Alert",
     "type":"object",
     "properties": {
          "Alert": {
               "type": "array",
               "items": {
                    "description": "status codes are ordered in the same
sequence with the DM commands in the Package#2",
                    "type": "object",
                    "properties": {
                         "AlertType": {
                              "type": "string"
                         },
                         "SourceURI": {
                              "type": "string"
                         },
                         "TargetURI": {
                              "type": "string"
                         },
                         "Mark": {
                              "type": "string"
```

```
                        },
                        "DataType": {
                             "type": "string"
                        },
                        "Data": {
                             "type": "object",
                             "description": "the format is out-of-scope of this
specification"
                        }
                   },
                   "required": ["AlertType"]
              }
         }
    }
}
```

This is an illustrative example of Generic Alert.

```
{
     "Alert": [
          {
               "AlertType": "urn:oma:at:scomo:1.0:OperationComplete",
               "SourceURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Download/Package1/Operations/DownloadInstall",
               "TargetURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Inventory/Deployed/Component1",
               "Mark": "warning",
               "DataType": "text/xml",
               "Data": {
                    "DM1x": "<ResultCode>1200</ResultCode>          <!-- SCOMO
Result Code --><Identifier>Component1ID</Identifier>"
               }
          },
          {
               "AlertType": "urn:oma:at:scomo:1.0:OperationComplete",
               "SourceURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Download/Package1/Operations/DownloadInstall",
               "TargetURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Inventory/Deployed/Component2",
               "Mark": "warning",
               "DataType": "text/xml",
               "Data": {
                    "DM1x": "<ResultCode>1200</ResultCode>          <!-- SCOMO
Result Code --><Identifier>Component1ID</Identifier>"
               }
          }
     ]
}
```

7.2.2 Other Format

The DM Server and the DM Client MAY implement other serialization format which are out of the scope of this specification.

8. Bootstrap

Bootstrap is the process of provisioning the DM Client to a state where it is able to initiate a DM Session to a new DM Server. The DM Clients that have already been bootstrapped can be further bootstrapped to enable them to initiate a DM Session to new DM Servers or may be rebootstrapped to update existing accounts.

The following bootstrap methods are supported:

- Factory Bootstrap: the DM Bootstrap is preloaded from factory
- Smartcard Bootstrap: the DM Client retrieves the DM Bootstrap from a Smartcard
- Client initiated bootstrap: the DM Client retrieves the DM Bootstrap from a DM Bootstrap Server, whose URL is known to the Device.
- Server initiated bootstrap: the DM Server pushes the DM Bootstrap to the DM Client upon discovering sufficient information about the device.

The DM Client MUST support at least one of these mechanisms.

When DM Bootstrap has been successfully installed in the DM Client, the DM Client MUST initiate a DM Session with the bootstrapped DM Server in providing an Alert with the AlertType “urn:oma:at:dm:2.0:BootstrapComplete” (see Appendix D).

8.1 Smartcard bootstrap

If the Device supports a Smartcard, the DM Client MUST support detection, retrieval, and processing of the DM Bootstrap from the Smartcard as described in Appendix G. The Device MAY include configurable security policy to disable Smartcard bootstrap functions. If the Smartcard bootstrap function is enabled (i.e. no security policy is implemented or security policy does not disable Smartcard bootstrap) and the Smartcard has not been rejected by the device (for example, because of a SIM-locking mechanism), the DM Client MUST retrieve the DM Bootstrap from the Smartcard when the Device is switched on and apply it.

The DM Client MUST check that the bootstrap data for all DM Servers previously bootstrapped from the Smartcard are still available from the Smartcard when the Device is switched on; if not, the information for any DM Servers that were previously bootstrapped from the Smartcard but are no longer stored on the Smartcard MUST be removed from the DM Tree.

According to the Bootstrap Format Section 8.5, the DM Bootstrap in Smartcard MUST be formatted and serialized as specified in Section 7.2.1.4. Between the Smartcard and the DM Client, the DM Bootstrap is conveyed in using JSON format with utf-8 encoding without BOM.

Due to the sensible nature of the DM Bootstrap, a secure channel SHOULD be established between the Smartcard and the DM Client. When such a secure channel is established between the Smartcard and the DM Client, this secure channel SHALL be based on [GPSCP03] procedure, mainly described in Appendix H.

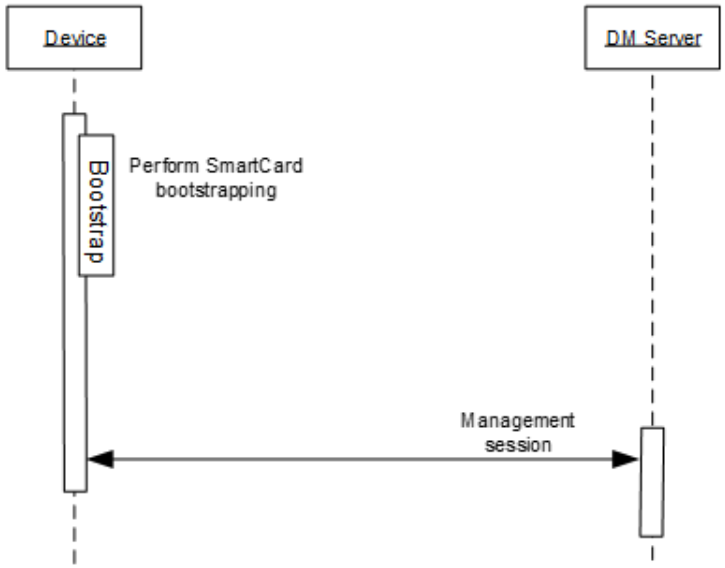


Figure 11: SmartCard bootstrap

8.2 Factory bootstrap

In this case all parameters to access the DM Server are already configured in the DM Client from factory. The mechanism of provisioning this configuration is out of the scope of this specification.

The DM Client SHOULD support this mechanism.

8.3 Client initiated bootstrap

In this scenario, the DM Client requests and retrieves the DM Bootstrap from a DM Server, whose URL (the format is defined in chapter 6.2) is known to the DM Client a priori, as shown in Figure 12.

The DM Client SHOULD support this mechanism.

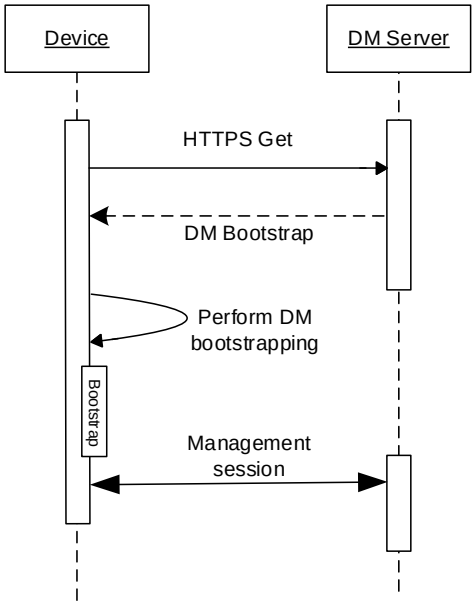


Figure 12: Client initiated bootstrap

8.4 Server initiated bootstrap

In this scenario, upon manufacturing a Device, when this one leaves the factory it is packaged and delivered in an empty state. Once a user acquires the Device and personalizes it (e.g. by inserting a Smartcard), the prerequisites for this process are in place. The problem is how to inform the DM Server of the identity, address or phone number of the Device. This can be addressed in many ways discussed below.

- It could be done at the point-of-sales where a sales system ties in with the management system and delivers the information.
- It could be done through a self-service web site where the user enters his/her own phone number and this information is delivered to the management system.
- It could be done by the network when the Device registers to the network for the first time. When this happens a trigger could be sent from the core network to the DM Server with the Device's number.
- It could be done with a voice prompt system where the user is prompted to key in his/her phone number.

Regardless of how the phone number or Device address reaches the DM Server, the DM Server is now in a position where it can send out a DM Bootstrap to the DM Client.

The DM Clients SHOULD accept DM Bootstrap only from trusted DM Servers, and the DM Server MUST send the DM Bootstrap to the DM Client using a secure channel.

The DM Server MUST support this mechanism. The DM Client SHOULD support this mechanism.

Figure 13 gives an overview of this scenario.

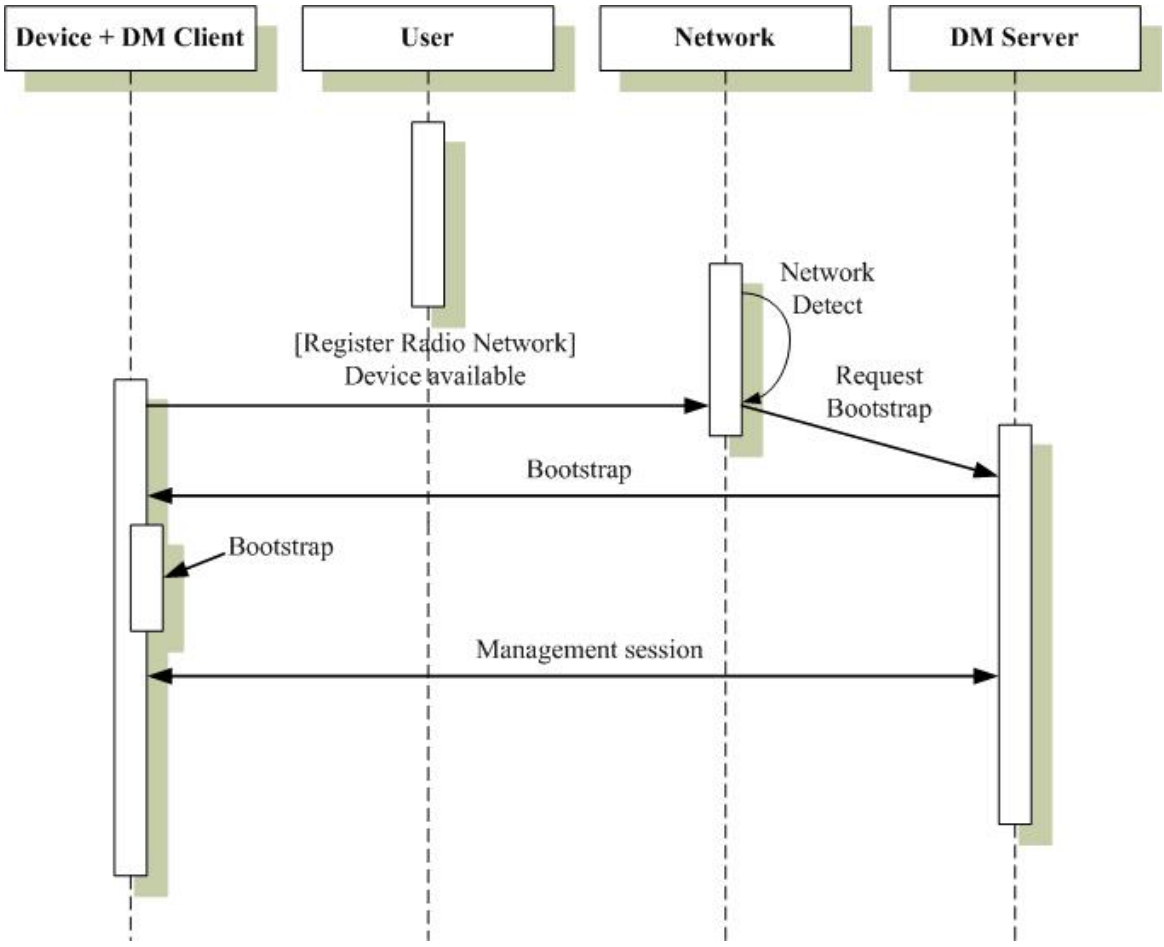


Figure 13: Server initiated bootstrap

The transmission of the DM Bootstrap MUST be secured by using one of security mechanisms described in the Section 9.3.

8.5 Bootstrap Format

The DM Bootstrap contains MO data which the DM Client MUST add or replace into the DM Tree, with the same behaviour of HGET DM Command without the ClientURI specified (see chapter 5.3.1). The MO data conveyed by DM Bootstrap MUST be formatted and serialized according to the rules for serializing any other Management Object (see chapter 7.2.1.4).

The DM Bootstrap normally conveyed the DMAcc MO, but any other MO may be included.

9. Protocol Security

9.1 Security for DM Notification

The level of the security, required for the DM Notification, could be various. This specification does not provide the full security suite to cover those various cases. The DM Notification specified in this specification provides the integrity and authentication based on the SHA256-DIGEST Option. The TIMESTAMP Option also can be used to prevent the reply attacks. However, this specification does not provide any means for guaranteeing the confidentiality for the DM Notification. In the case that the DM Notification does not satisfy the security level required for the deployment, the underlying transport layer security MUST be used.

To use the SHA256-DIGEST Option, credentials MUST be provisioned at the device, and the *<x>/Credentials/Noti* sub-tree in the DM Account MO MAY be used, but other approaches are not precluded.

9.2 Security for DM Session

The DM Protocol does not provide any security mechanisms to guarantee the integrity and confidentiality of DM Session and for the mutual authentication between the DM Client and the DM Server, but relies on the transport layer security mechanisms.

SSL [SSL] and TLS are protocols defined by IETF allowing transmission of data over insecure network such as the Internet. Those protocols are layered between the application protocol layers (e.g. HTTP) and TCP/IP. HTTPS means HTTP over SSL/TLS: even if SSL is used by old systems, the usage of TLS is recommended as SSL 3.0 is weaker than TLS 1.0.

TLS provides mechanisms to establish a secure session between two entities. Here the DM Server and the DM Client who can be named as peers.

Those mechanisms can protect the peer against attacks such as man-in-the-middle, eavesdropping and others defined in the [ISO29115] specification. The protection depends on the level of assurance the service provider (e.g. an MNO) wants to give.

Authentication (mutual or not), authorization, confidentiality and integrity are the main functions that these mechanisms provide to secure the peers.

9.2.1 Authentication

TLS support the use of several authentication mechanisms to allow the client to authenticate the server and vice-versa.

In the context of this specification:

- the DM Server MUST support Basic Authentication Schema [RFC2617], Digest Authentication Schema [RFC2617] and X.509 certificate [RFC5280] and [RFC6125];
- the DM Client MUST support Basic Authentication Schema [RFC2617] and X.509 certificate [RFC5280] and [RFC6125]; it SHOULD support Digest Authentication Schema [RFC2617].

The Authentication and Authorization control appear during the first step of the TLS handshake.

The necessary credentials SHALL be provisioned at the device (see section 8), and the *<x>/Auth/Trsp* sub-tree in the DM Account MO SHOULD be used.

9.2.2 Confidentiality and integrity

Confidentiality means that the application data are encrypted prior to be transmitted over TCP. Two main types of encryption exist: symmetric key also known as shared secret key and asymmetric key also known as public key or public-private key. TLS use both encryption methods.

The method and the Cipher Suite to be used by the peer are the result of the TLS handshake.

For more details on TLS refers to TLS 1.0 [RFC4279], TLS 1.1 [RFC4346] and TLS 1.2 [RFC5246].

For the Cipher Suites defined in TLS, refers to TLS Cipher Suite registry http://www.iana.org/assignments/tls-parameters/tls-parameters.xml

Mechanism defined in [RFC2817] and [RFC2818] SHOULD be used to upgrade the level of security for HTTP over TLS (e.g.: an HTTP client can request the use of TLS 1.2 and can refuse the connection if the server doesn't support TLS 1.2).

If X509 certificate mode is not supported, Pre-Shared Keys mode (PSK-TLS) can be used.

Specifically:

- The DM Server SHALL support X509 certificate and PSK-TLS [RFC4279]
- The DM client SHALL support at least one of the following modes:
  - X509 certificate
  - PSK-TLS
- The DM Server and the DM Client SHALL support TLS 1.1 [RFC4346]
- The DM Server and the DM Client SHOULD support TLS 1.0 [RFC4279] or SHOULD support TLS 1.2 [RFC5246]
- The DM Client SHALL check that the DM Server is using TLS1.0 [RFC4279] or TLS 1.1 [RFC4346] or TLS 1.2 [RFC5246] before accepting the session establishment
- A session SHALL NOT take place over mechanism weaker than TLS1.0 [RFC4279]
- The DM Server SHALL support the following Cipher Suites when using an PSK-TLS mode:
  - TLS_PSK_WITH_AES_128_GCM_SHA256
  - TLS_DHE_PSK_WITH_AES_128_GCM_SHA256
  - TLS_RSA_PSK_WITH_AES_128_GCM_SHA256
  - TLS_PSK_WITH_AES_128_CBC_SHA256
  - TLS_DHE_PSK_WITH_AES_128_CBC_SHA256
  - TLS_RSA_PSK_WITH_AES_128_CBC_SHA256
  - TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256
- The DM Client SHALL support at least one of the following Cipher Suites when using an PSK-TLS mode:
  - TLS_PSK_WITH_AES_128_GCM_SHA256
  - TLS_DHE_PSK_WITH_AES_128_GCM_SHA256
  - TLS_RSA_PSK_WITH_AES_128_GCM_SHA256
  - TLS_PSK_WITH_AES_128_CBC_SHA256
  - TLS_DHE_PSK_WITH_AES_128_CBC_SHA256
  - TLS_RSA_PSK_WITH_AES_128_CBC_SHA256
  - TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256
- The DM Server SHALL support the following Cipher Suites when using X509 mode:
  - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
  - TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
  - TLS_RSA_WITH_AES_128_GCM_SHA256
  - TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256

  - TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
  - TLS_RSA_WITH_AES_128_CBC_SHA256
- The DM Client SHALL support at least one of the following Cipher Suites when using X509 mode:
  - TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
  - TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
  - TLS_RSA_WITH_AES_128_GCM_SHA256
  - TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
  - TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
  - TLS_RSA_WITH_AES_128_CBC_SHA256
- When an ECDHE or ECDSA algorithm is used, the minimum key length SHALL be at least 256 bits

9.3 Security for Bootstrap

Bootstrapping is a sensitive process that may involve communication between two parties without any previous relationship or knowledge about each other. In this context, security is very important. The receiver of a bootstrap message needs to know that the information originates from the correct source and that it has not been tampered with en-route. The sender also wants end-to-end confidentiality to prevent impersonation by eavesdroppers who could see the contents of the bootstrap message containing credentials to access the DM Server.

It is important that DM Clients accept bootstrapping commands only from authorized DM Servers.

9.3.1 Bootstrap via DM

9.3.1.1 Transports

The Bootstrap message MUST be sent to the DM Client only if a secure transport is available.

Transport specific security MUST be used such as TLS.

9.3.1.2 Smartcard

The Smartcard provides secure mechanisms to transport and store the Bootstrap message information.

For specific specification of a Smartcard, please refer to [OMADICT].

Bootstrap data MAY be stored in the Smartcard. The behaviour of a DM Client regarding bootstrap data is specified in the section 8.1.

9.3.1.3 Network Dependent Security

This method applies only to 3GPP Networks and devices that support GBA Push.

It is assumed that the DM Server has access to both a DM Bootstrap Integrity Key (DMBIK) and a DM Bootstrap Encryption Key (DMBEK) which have been derived from the long-term secret that is shared between the device Smartcard and the network using the 3GPP Generic Bootstrapping Architecture (GBA) Push specifications [TS33.223].

The GBA Push procedures MUST be executed prior to sending the bootstrap message itself in order for terminal and network to agree on the DMBIK and DMBEK that SHALL be used to protect the bootstrap message. For more information on how a DM Server can interact with GBA Push see [DM_1.3].

The security method and combined integrity and confidentiality are passed as parameters to the content type in the format like this:

Content-Type: MIME type; SEC=type

Where:

MIME type is [TBD]

SEC = “3GPP_GBA”

GBA Push allows the generation of a so called Ks_(ext/int)_NAF shared secret both in the network and in the device. From this master key Ks_(ext/int)_NAF, two shared secrets are generated: the DMBIK and the DMBEK.

This 3GPP_GBA method requires:

- The NAF_Id SHALL be constructed using as FQDN the DM Server FQDN and as GBA Ua security protocol identifier the one defined for DM in Open Mobile Naming Authority (OMNA).
- An integrity and confidentiality protected bootstrap message using DMBIK and DMBEK shared secrets and derived from the Ks_(ext/int)_NAF using the GBA key derivation function as follows :
    - FC = 0x01
    - For DMBIK: P0 = "dmbik" (i.e. 0x64 0x6D 0x2D 0x62 0x69 0x6B)
    - For DMBEK: P0 = "dmbek" (i.e. 0x64 0x6D 0x2D 0x62 0x65 0x6B)
    - L0 = length of P0 is 6 octets (i.e. 0x00 0x06).

    See Annex B of [TS33.220] for notation style and how parameters are used.

    The Key to be used in key derivation SHALL be:

    - Ks_(ext/int)_NAF

    In summary, the DMBIK and DMBEK SHALL be derived from the Ks_(ext/int)_NAF and static strings "dmbik" and "dmbek" respectively as follows:

    - DMBIK = KDF (Ks_(ext/int)_NAF, "dmbik")
    - DMBEK = KDF (Ks_(ext/int)_NAF, "dmbek")

The protocol used to send the bootstrap message MUST be capable of transporting the protected OMA DM bootstrap package.

9.3.2 Authentication and Authorization

During the Bootstrap procedure, the DM Client and the DM Server MUST perform a Mutual Authentication. This section uses the same mechanism specified in Section 9.2.1 for securing a non bootstrap DM Session. The only difference is that the key materials used for securing the transmission of the Bootstrap message are stored under the *<x>/Auth/Bootstrap* node of DM Account MO and cannot be changed by any other DM Server not acting as the bootstrap server.

10.Delegation Access Control Mechanism

The Access Control Mechanism ensures that only authorized DM Servers can invoke DM commands on the MO instances. The Access Control Mechanism is designed with the following principles:

- The DM Bootstrap contains a list of MOIDs (the *Permissions/<x>/MOID* node in DM Account MO) requested to be managed by the DM Server. Accepting this DM Bootstrap does not mean that the DM Server can immediately manage the MO instances for the provisioned MOIDs. To manage a specific MO instance for the provisioned MOID, the DM Server MUST get permissions for the MO instance. The DM Client MUST ensure that the DM Server SHALL NOT manage Management Objects whose MOID is not provisioned at the bootstrap

- The creator (i.e., the DM Server, an application running locally on the Device) of a MO instance automatically gets the exclusive full permissions for the new MO instance by default

- The access rights are assigned on MO instance base and enforce the authorization of the execution of DM Commands targeting nodes contained in that instance. It is possible, however, to assign access rights to a part of MO instance (e.g., a sub-tree or a leaf node) overwriting the access rights assigned to the MO instance; this is necessary to guarantee interwork with DM 1.x ACL mechanism (refer to the section 13.2 for details)

The DM Client MUST use the Delegation Access Control MO (see section 12.2.1) to expose to the DM Server the MO instance the DM Server can have access to and the relate access rights.

The DM Server MUST support the Access Control Mechanism.

The DM Client SHOULD support the Access Control Mechanism; if the DM Client does not support the Access Control Mechanism, the DM Client MUST grant full permissions to all MO instances to every bootstrapped DM Server.

11.Management Object Cache Mechanism

When the DM Server retrieves a MO data (an entire MO or a part of a MO) from the DM Client, the Management Object Cache mechanism can be used to reduce network traffic and the response latency. The DM Client can store locally a copy (i.e. cache) of the MO data, and the subsequent requests for the same MO can refer to the cached data if certain conditions (the cache hit) are met.

The DM Server SHOULD support this mechanism. The DM Client MAY support this mechanism.

11.1 Cache Validator

A cache validator is an entity which gives the freshness information for the local cache. Typical examples of cache validator are timestamp or opaque identifier like the HTTP ETag header [RFC2616].

Only MO instances can be cached and these MO instances are called “cacheable MO Instances”: the process of selection of the cacheable MO Instances is out-of-scope of this specification. For each cacheable MO instance, the DM Client MUST assign a “cache validator” which MUST be updated every time a change in the MO instance occurs.

The cache validation is the process of checking whether the cached copy is valid or stale: the cache validation MUST return true if the whole MO instance has not been modified compared to the local cache; otherwise, the cache validation MUST return false.

Although the cache validator is assigned to a cacheable MO instance, the cache validator can be used for all read operations (i.e., GET/HPUT/HPOST) targeting any node in a cacheable MO instance. For example, if the ClientURI of a GET command targets a leaf node in the cacheable MO instance, the cache validator for the MO instance can be provided in the cv field, and the DM Server will receive "304 Not Modified" if the whole MO instance is not modified.

The cv field in the ClientURI provides the cache validator for the cacheable MO instance.

11.2 Request and Response with Cache

The following describes the DM Server part of Cache Mechanism flow.

Step 1: The DM Server wants to request a MO data identified by a ClientURI.

Step 2a: If the DM Server knows the cache validator for the MO instance containing the node targeted by the ClientURI, the DM Server SHOULD add the cv field to the ClientURI. The cv field in the ClientURI provides the cache validator for the MO instance.

Step 2b: If the DM Server doesn’t know the cache validator for the MO instance, then the DM Server SHALL NOT include the cv field in the ClientURI.

The following describes the DM Client part of Cache Mechanism flow.

Step 1: The DM Client receives a DM command requesting a MO data identified by a ClientURI.

Step 2a: If the ClientURI doesn’t have the cv field, the DM Client MUST return the requested MO data. If the ClientURI targets a cacheable MO instance (i.e., the ClientURI targets the root node of the MO instance), the DM Client SHOULD return also the cache validator for that MO instance.

Step 2b: If the ClientURI has the cv field, the DM Client MUST run the cache validation process:

- if the cache validation process returns true, the DM Client MUST return "304 Not Modified", and SHALL NOT send the requested MO data
- if the cache validation process returns false, the DM Client MUST return the requested MO data. If the ClientURI targets a cacheable MO instance (i.e., the ClientURI targets the root node of the MO instance), the DM Client SHOULD return also the cache validator for that MO instance.

To return the cache validator in the Step2a and Step 2b, the Management Object JSON object specified in the section 7.2.1.4 has the name/value pair where the name is set to "CV", and the value is set to the cache validator accordingly.

12.DM 2.0 Standard Management Objects

12.1 DevInfo Management Object

The DM Server and the DM Client MUST support this MO.

The following figure shows an overview of the MO.

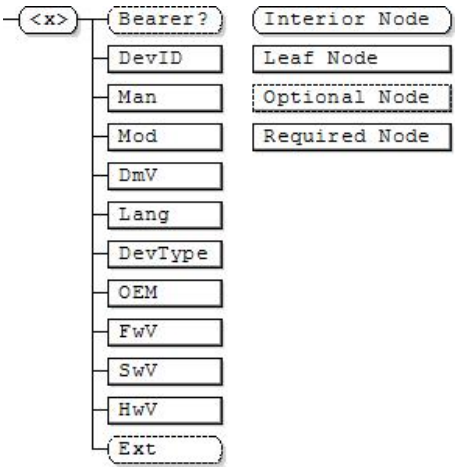


Figure 14: The DevInfo Management Object

<x>

Status	Occurrence	Format	Min. Access Types
Required	One	node	Get

This interior node is the root node for the DevInfo MO. The MOID for the DevInfo MO MUST be: "urn:oma:mo:oma-dm-devinfo:1.2".

<x>/Bearer

Status	Occurrence	Format	Min. Access Types
Optional	ZeroOrOne	node	Get

An optional, interior node in which items related to the bearer (CDMA, etc.) are stored. Use of this sub tree can be mandated by other standards.

<x>/DevID

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

This leaf node specifies the global unique identifier for the device. The value of this node MUST be either an absolute or a relative URI or a well-known URN. Possible formats for this node are listed in the below table, but other formats are not precluded.

Type	Descriptions
IMEI URN	Identify International Mobile Equipment Identifiers [3GPP-TS_23.003]. The IMEI URN specifies a valid, 15 digit IMEI. The format of the URN is IMEI:################
ESN URN	Identify an Electronic Serial Number. The ESN specifies a valid, 8 digit ESN. The format of the URN is ESN:########
MEID URN	Identify a Mobile Equipment Identifier. The MEID URN specifies a valid, 14 digit MEID. The format of the URN is MEID:##############
UUID URN	Identify an Universally Unique IDentifier (UUID). The UUID specifies a valid, hex digit character string as defined in [RFC4122]. The format of the URN is UUID:########-####-####-############

<x>/Man

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

The manufacturer identifier (manufacturer specified string).

<x>/Mod

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

A model identifier (manufacturer specified string).

<x>/DmV

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

OMA device management client version identifier (manufacturer specified string).

<x>/Lang

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

The current language setting of the device. The syntax of the language tags and their use are defined in [RFC1766]. Language codes are defined by ISO in the standard [ISO639-2].

<x>/DevType

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

Device type, for example PDA, pager, or phone (manufacturer specified string).

<x>/OEM

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

Original Equipment Manufacturer of the device (manufacturer specified string).

<x>/FwV

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

Firmware version of the device (manufacturer specified string).

<x>/SwV

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

Software version of the device (manufacturer specified string).

<x>/HwV

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

Hardware version of the device (manufacturer specified string).

<x>/Ext

Status	Occurrence	Format	Min. Access Types
Optional	One	node	Get

This interior node is for vendor-specific extensions to store the device related information.

12.2 DM Account Management Object

The DM Server and the DM Client MUST support this MO.

The following figure shows an overview of the MO.

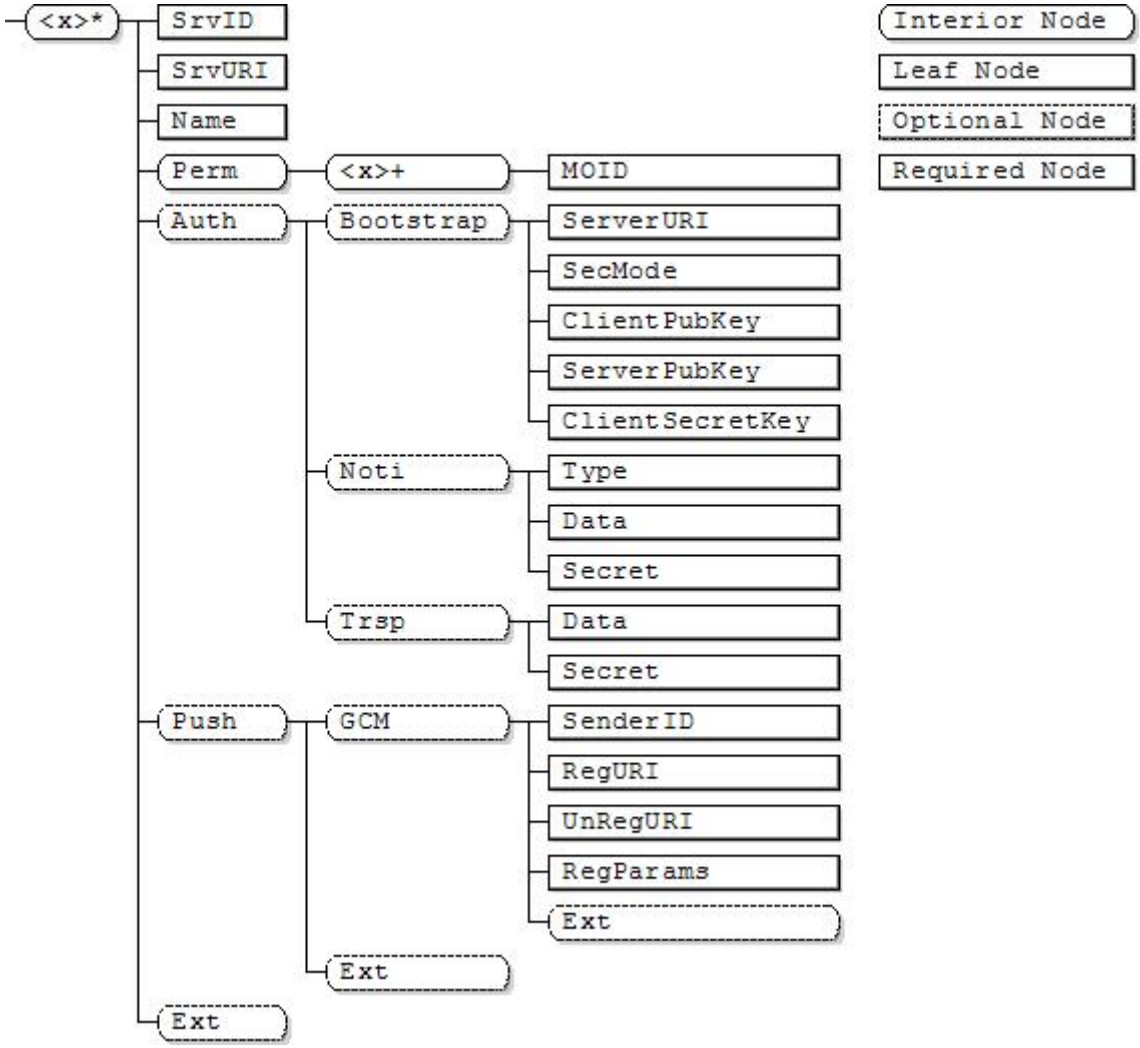


Figure 15: The DM Account Management Object

<x>

Status	Occurrence	Format	Min. Access Types
Required	ZeroOrMore	node	Get

This interior node acts as a placeholder for one or more instances of this object. Management Object Identifier for this management object MUST be: "urn:oma:mo:oma-dm-dmacc:1.2".

<x>/SrvID

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

This node specifies a server identifier for the management server used in the management session. This identifier MUST be unique within the DM Client.

<x>/SrvURI

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

This node specifies the DM Server endpoint the DM Client MUST connect in order to start the DM Session. It MUST be encoded as URI [RFC3986].

<x>/Name

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

This node specifies user displayable name for the management server.

<x>/Perm

Status	Occurrence	Format	Min. Access Types
Required	One	node	No Get

This interior node is the parent node for the permission related information that is provided during the bootstrap only. The actors such as a user MAY reject the bootstrap message based on the information. This node and all child nodes MUST NOT be exposed in the DM Client.

<x>/Perm/<x>

Status	Occurrence	Format	Min. Access Types
Required	OneOrMore	node	No Get

This node groups the permission related information for a specific MOID.

<x>/Perm/<x>/MOID

Status	Occurrence	Format	Min. Access Types
Required	One	chr	No Get

This node specifies the MOID for this group. Based on this node, the MOS (Management Object Supported) information is exposed to the DM Server as specified in the section 5.2.2, and the DM Server gets the access rights for MO instances as specified in the section 12.2.1.
The wildcard "*" MAY be used for the value of this node, and it MUST be interpreted for all MOIDs that the DM Client supports.

<x>/Auth

Status	Occurrence	Format	Min. Access Types
Optional	One	node	Get

This interior node is the parent node for authentication managed by the DM Server.

<x>/Auth/Bootstrap

Status	Occurrence	Format	Min. Access Types
Optional	One	node	Get

This interior node is the parent node where are stored the keying materials and secret data used to establish a SSL/TLS session with a DM server acting as a bootstrap server. If this node is present:

- the DM Client MUST use the data specified in the child nodes for executing all the necessary functions to ensure the DM Server Authentication,
- the DM Server SHOULD use the data specified in the child nodes for executing all the necessary functions to ensure the DM Client Authentication.
- the DM client MUST use the data specified in the child nodes for executing Authorization, Confidentiality and Integrity of the peer DM Server/DM Client as specified in section 9.3.

If this node is not present, the credentials for the Bootstrap security MUST be delivered via another mechanism out-of-scope of this specification.

<x>/Auth/Bootstrap/ServerURI

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

This node specified the URI of the DM Server acting as Bootstrap server. For example: https://www.bsdmserver.com:443 or “111.222.333.444” (IPv4 address)

<x>/Auth/Bootstrap/SecMode

Status	Occurrence	Format	Min. Access Types
Required	One	bin	Get

This node specifies the security mode of SSL/TLS. For example, PSK mode, X.509 mode.

<x>/Auth/Bootstrap/ClientPubKey

Status	Occurrence	Format	Min. Access Types
Required	One	bin	No Get

This interior is the placeholder of the DM Client Certificate (X.509 Certificate mode) or PSK Identity (PSK mode). The format of this node is defined in Section 12.2.1.

<x>/Auth/Bootstrap/ServerPubKey

Status	Occurrence	Format	Min. Access Types
Required	One	bin	No Get

This interior is the placeholder of the DM Server Certificate (X.509 Certificate mode) or PSK Identity (PSK mode). The format of this node is defined in Section 12.2.1.

<x>/Auth/Bootstrap/ClientSecretKey

Status	Occurrence	Format	Min. Access Types
Required	One	bin	No Get

This interior is the placeholder of the DM Client secret key (PreShared Key in PSK mode or Private Key in X.509 Mode). The format of this node is defined in Section 12.2.1.

<x>/Auth/Noti

Status	Occurrence	Format	Min. Access Types
Optional	One	node	Get

This interior node is the parent node for credentials used for DM Notification. If this interior node is presented and the SHA256-DIGEST Option is specified in the DM Notification, then the DM Client MUST authenticate the DM Notification using the SHA256-DIGEST Option.

<x>/Auth/Noti/Type

Status	Occurrence	Format	Min. Access Types
Required	One	int	Get

This node specifies the authentication type. The Value of this node MUST be one of the following:

Valid Value	Descriptions
0	SHA256 digest as specified in [RFC6234]

<x>/Auth/Noti/Data

Status	Occurrence	Format	Min. Access Types
Required	One	bin	Get

This node specifies the authentication data related to the authentication type indicated by the *<x>/Auth/Noti/Type* node.

<x>/Auth/Noti/Secret

Status	Occurrence	Format	Min. Access Types
Required	One	bin	Get

This node specifies the authentication secret related to the authentication type indicated by the *<x>/Auth/Noti/Type* node.

<x>/Auth/Trsp

Status	Occurrence	Format	Min. Access Types
Optional	One	node	Get

This interior node is the parent node for credentials used for securing the DM session by the transport layer. If this node is present, the DM Client MUST use credentials specified in the child nodes for transport layer security mechanism. If this node is not present, the credentials for the transport layer security MUST be delivered via the out-of-scope mechanisms.

<x>/Auth/Trsp/Data

Status	Occurrence	Format	Min. Access Types
Required	One	bin	Get

This node specifies the authentication data used for the transport layer security. For example, this node can store the certificate or the public key for the DM Client, which can be used for the transport security.

<x>/Auth/Trsp/Secret

Status	Occurrence	Format	Min. Access Types
Required	One	bin	No Get

This node specifies the authentication secret used for the transport layer security. For example, this node can store the private key for the DM Client, which can be used for the transport security.

<x>/Push

Status	Occurrence	Format	Min. Access Types
Optional	One	node	Get

This node is a placeholder node for platform or vendor specific push mechanism.

<x>/Push/GCM

Status	Occurrence	Format	Min. Access Types
Optional	One	node	Get

This node is a placeholder node for GCM push mechanism configuration (see Appendix E).

<x>/Push/GCM/SenderID

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

This node contains the sender ID [GCM] to identify the service.

<x>/Push/GCM/RegURI

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

This node contains the URI for where the DM Client MUST register itself after receiving a Register [GCM] event.

<x>/Push/GCM/UnRegURI

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

This node contains the URI for where the DM Client unregisters itself once received an UnRegister [GCM] event.

<x>/Push/GCM/RegParams

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

This node contains additional registrations parameters that the DM Client MUST use during register and unregister. If this string exists, it MUST be encoded as the URI query component [RFC3986].

<x>/Push/GCM/Ext

Status	Occurrence	Format	Min. Access Types
Optional	One	node	Get

This interior node is for vendor-specific extensions to store the device related information.

<x>/Push/Ext

Status	Occurrence	Format	Min. Access Types
Optional	One	node	Get

This interior node is for vendor-specific extensions to store the device related information.

<x>/Ext

Status	Occurrence	Format	Min. Access Types
Optional	One	node	Get

This interior node is for vendor-specific extensions to store the device related information.

12.2.1 Security Key and Identity Format

This section defines the format of the Secret Key, Public Key and Identity nodes of the DM Account Management Object. These nodes are used to configure the security mode and keying materials that a DM Client uses with a DM Server acting as a bootstrap server. Those nodes are configured in the DM Client using one of the Bootstrap mechanisms described in Section 8. The use of this keying material for each security mode is defined in Section 9.2.

12.2.1.1 Pre-Shared Key (PSK) Mode

The PSK is a binary shared secret key between the Client and Server of the appropriate length for the Cipher Suite used [RFC4279]. This key is composed of a sequence of binary bytes in the ClientSecretKey node.

The corresponding PSK Identity for this PSK is stored in the Client-PublicKey-PSKIdentity node. The PSK Identity is simply stored as a UTF-8 String as per [RFC4279]. Clients and Servers MUST support a PSK Identity of at least 128 bytes in length as required by [RFC4279].

12.2.1.2 Certificate Mode

The Certificate mode requires an X.509v3 Certificate along with a matching private key. The private key is stored in the ClientecretKey node. The Certificate is simply represented as binary X.509v3 in the value of the Client-PublicKey-PSKIdentity node.

12.3 Delegation Access Control MO

The Delegation Access Control MO is used by the DM Client to expose the permission information to the DM Server. It is used by the “Access Control Mechanism” specified in section 10.

The DM Server MUST support this MO. If the DM Client support the “Access Control Mechanism” , MUST support this MO.

The following figure shows an overview of the Management Object.

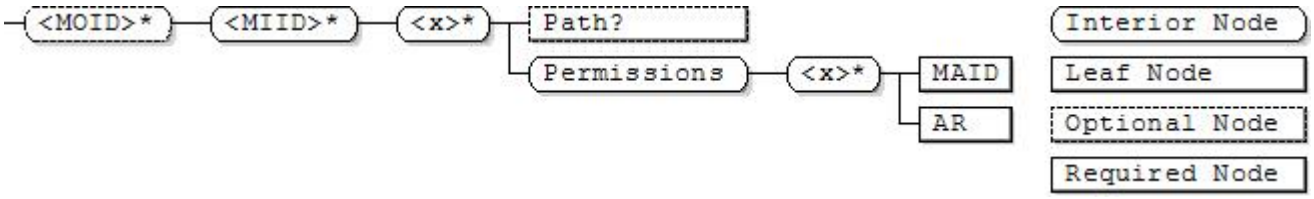


Figure 16: Delegation Access Control MO

<MOID>

Status	Occurrence	Format	Min. Access Types
Optional	ZeroOrMore	node	Get

This interior node is the placeholder for permissions regarding all MO instances for a particular MOID specified by the name of this node. The name of this node MUST be the MOID.

The MOID for this Management Object MUST be: "urn:oma:mo:oma-dm-dacmo:1.0".

<MOID>/<MIID>

Status	Occurrence	Format	Min. Access Types
Required	ZeroOrMore	node	Get

This interior node is the placeholder for permissions regarding an MO instance identified by the MOID as specified by the <*MOID*> node and the MIID as specified by the name of this node. The name of this node MUST be the MIID of the MO instance.

<MOID>/<MIID>/<x>

Status	Occurrence	Format	Min. Access Types
Required	One	node	Get

This interior node is the placeholder for permissions regarding an MO instance.

<MOID>/<MIID>/<x>/Path

Status	Occurrence	Format	Min. Access Types
Optional	ZeroOrOne	chr	Get

This node specifies the path to a certain node in the MO instance. The format of this node MUST conform to the path-from-moroot component of the ClientURI as specified in the section 6.1.1.2. Hence this node describes the relative path from the MO instance root node. The permissions specified by the *<MOID>/<MIID>/<x>/Permissions* node are applied to the sub-tree addressed by this node unless the permissions are overwritten at the descendent node.

If this node is not present, the default path "/" MUST be assumed.

This node is useful to interwork with the DM 1.3 ACL mechanism and the details are explained in the section 13.2.

<MOID>/<MIID>/<x>/Permissions

Status	Occurrence	Format	Min. Access Types
Required	One	node	Get

This interior node groups permissions regarding the sub-tree addressed by the *<MOID>/<MIID>/<x>/Path* node in the MO instance. If the *<MOID>/<MIID>/<x>/Path* node is not present, the default path "/" MUST be assumed.

<MOID>/<MIID>/<x>/Permissions/<x>

Status	Occurrence	Format	Min. Access Types
Required	One	node	Get

This interior node is the placeholder for permissions regarding the sub-tree addressed by the *<MOID>/<MIID>/<x>/Path* node in the MO instance.

<MOID>/<MIID>/<x>/Permissions/<x>/MAID

Status	Occurrence	Format	Min. Access Types
Required	One	chr	Get

This node specifies the Management Authority that can be the DM Server or the local application running in the device. The value of this node MUST contain either the server identifier (i.e., one of values from the *<x>/ServerID* in the DM Account MO) or the application identifier. The value MUST be encoded as a URN with the prefix "serverid:" for a server identifier or "appid:" for an application identifier. The format of the application identifier might be platform-dependent, and out-of-scope of this specification.

The wildcard "*" MAY be used for the value of this node, and it MUST mean every DM Server and every local application.

The Management Authority identified by this node has the permissions specified by the <MOID>/<MIID>/<x>/Permissions/<x>/AR node for the sub-tree addressed by the <MOID>/<MIID>/<x>/Path node.

<MOID>/<MIID>/<x>/Permissions/<x>/AR

Status	Occurrence	Format	Min. Access Types
Required	One	int	Get

This node specifies the access rights that the Management Authority identified by the *<MOID>/<MIID>/<x>/Permissions/<x>/MAID* node has for the sub-tree addressed by the *<MOID>/<MIID>/<x>/Path* node.

If a Management Authority has a Delegate access right, then the Management Authority SHOULD be able to change the access rights for the sub-tree addressed by the *<MOID>/<MIID>/<x>/Path* node unless the permissions are overwritten at the descendent node.

Even if the Management Authority (the delegating MA) has the Delegate access rights, the DM Client MUST reject the delegation request if the delegated Management Authority does not provide the MOID of the MO instance during the bootstrap.

The value of this node is a summary of the access rights value in this table:

Logical Operations	Commands for the Logical Operation	Value
Read	GET/HPUT/HPOST	1
Write	HGET/DELETE	2
Execute	EXEC	4
Delegate	HGET to modify the access rights of the sub-tree addressed by the *<MOID>/<MIID>/<x>/Path* node	8

12.3.1 Examples for this Management Object

In this section, illustrative examples for this Management Object are given, and the permissions are controlled only for the MO instances. Hence in this example, the *<MOID>/<MIID>/<x>/Path* node is not present, which means all specified permissions are for the MO instances, not for the part of the MO instance. Other examples to use the *<MOID>/<MIID>/<x>/Path* node are shown in the section 13.2.

For the DM Account MO, three DM Servers have been bootstrapped, and three MO instances of the DM Account MO are created in the device with the MIIDs; "dms1", "dms2" and "dms3". For the DevInfo MO, there exists only one MO instance in the device. Below shows the configurations of the Delegation Access Control MO for this case:

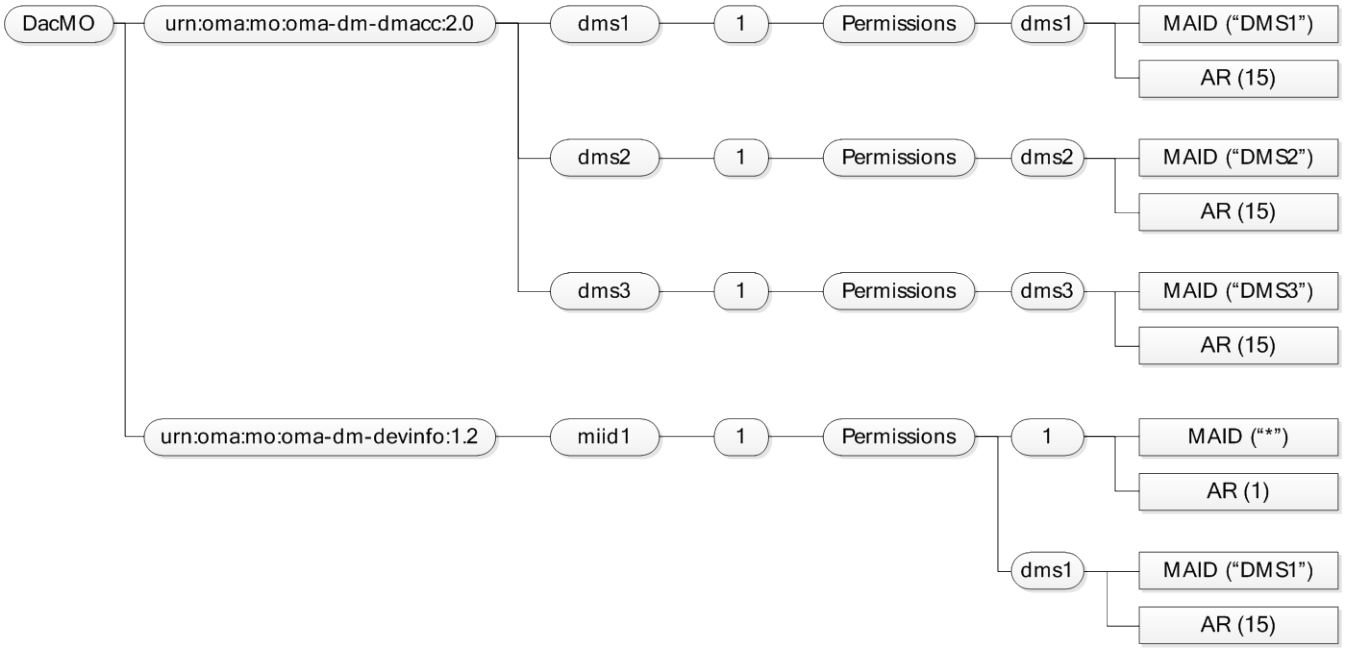


Figure 17: Examples for Delegation Access Control MO

12.4 Session Information Management Object

Session Information MO exposes the DM session information to the DM Server. The DM Server can request to include this MO in the Package#1 by using the REQ-MO Option in the Notification.

The DM Server and the DM Client MUST support this MO.

The pictorial description for this MO is as follows:

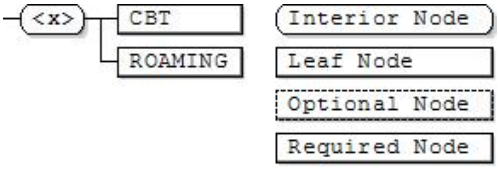


Figure 18: Session Information Management Object

<x>

Status	Tree Occurrence	Format	Min. Access Types
Required	One	node	Get, No Replace

This interior node is the root node for the Session Information Management Object. The MOID for this MO MUST be: "urn:oma:mo:oma-sessioninfomo:1.0".

<x>/CBT

Status	Tree Occurrence	Format	Min. Access Types
Required	One	int	Get, No Replace

This node provides bearer type information over which the DM session is currently being carried. The content of this node is an integer with the value in range from 0 to 255, and currently the following values are allocated for different bearer types. For the bearer types not covered in this version of specification the value '0' (Other Bearer Type) MUST be used.

Value	Name	Description
0	OTHERS/UNKNOWN	Other bearer types not covered in this list or the bearer type is unknown
1	MOBILE	The bearer type is mobile (e.g., 3GPP, 3GPP2, WiMAX, etc.)
2	WIRELESS	The bearer type is wireless (e.g., WLAN, Bluetooth, etc.)
3	WIRELINE	To indicate the bearer type is wireline (e.g., Ethernet, DSL, etc.)

<x>/ROAMING

Status	Tree Occurrence	Format	Min. Access Types
Required	One	int	Get, No Replace

This node indicated the roaming status for the current DM session if the value of the *<x>/CBT* node is "MOBILE"; it is undefined in other cases.

The following values are valid:

Value	Description
0	Current DM session is not over a roaming connection.
1	Current DM session is over a roaming connection
2	It is unknown if the current DM Session is over a roaming connection.

13.DM 1.x Interworking Issues

13.1 DM 1.x Generic Alert Interworking (Informative)

OMA DM 1.x Protocol specifies the Generic Alert mainly for two purposes; the asynchronous reporting and Client Initiated Alert. DM 2.0 Generic Alert can be used for those two purposes as well, and is backward compatible with the DM 1.x Generic Alert only in the functional level.

DM 2.0 Generic Alert properties (i.e., AlertType, SourceURI, TargetURI, Mark, DataType and Data) can be used (see the section 5.4) to delivery the DM 1.x Generic Alert. One difference is that, for the asynchronous reporting, the correlator in DM 1.x Generic Alert is not used in DM 2.0 since on receiving the asynchronous reporting the DM Server can find out the DM command that triggers the asynchronous reporting based on the SourceURI property. The interworking issues mainly reside in the deliver of the Generic Alert data since the Generic Alert data does not have a fixed format. Each specification for Management Objects defines the format of the Generic Alert data; mostly in the form of xml. These MO-specific Generic Alert data MAY be converted to JSON format since the conversion from the xml to JSON is obvious. For example, the SCOMO [SCOMO] shows the following example for the asynchronous reporting:

```
<Alert>
     <CmdID>2</CmdID>
     <Data>1226</Data>                                  <!-- Generic Alert -->
     <Correlator>correlator1</Correlator>
     <Item>
          <Source><LocURI>
               ./SCM/Inventory/Delivered/Package456/Operations/Install
          </LocURI></Source>
          <Target><LocURI>./SCM/Inventory/Deployed/Component1</LocURI></Target>
          <Meta>
               <Type xmlns="syncml:metinf">
                    urn:oma:at:scomo:1.0:OperationComplete
               </Type>
               <Format xmlns="syncml:metinf">xml</Format>
               <Mark xmlns="syncml:metinf">warning</Mark>
          </Meta>
          <Data>
               <![CDATA[
                    <ResultCode>1200</ResultCode> <!-- SCOMO Result Code -->
                    <Identifier>Component1ID</Identifier>
               ]]>
          </Data>
     </Item>
     <Item>
          <Source><LocURI>
               ./SCM/Inventory/Delivered/Package456/Operations/Install
          </LocURI></Source>
          <Target><LocURI>./SCM/Inventory/Deployed/Component2</LocURI></Target>
          <Meta>
               <Type xmlns="syncml:metinf">
                    urn:oma:at:scomo:1.0:OperationComplete
               </Type>
               <Format xmlns="syncml:metinf">xml</Format>
               <Mark xmlns="syncml:metinf">warning</Mark>
          </Meta>
          <Data>
               <![CDATA[
                    <ResultCode>1200</ResultCode> <!-- SCOMO Result Code -->
                    <Identifier>Component2ID</Identifier>
               ]]>
```

```
          </Data>
     </Item>
</Alert>
```

Above DM 1.x Generic Alert MAY be represented as the DM 2.0 Generic Alert by converting Generic Alert data to JSON object. The conversion rules are out-of-scope of this specification, and one example can be as follows:

```
{
     "Alert": [
          {
               "AlertType": "urn:oma:at:scomo:1.0:OperationComplete",
               "SourceURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Download/Package1/Operations/DownloadInstall",
               "TargetURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Inventory/Deployed/Component1",
               "Mark": "warning",
               "DataType": "application/json",
               "Data": {
                    "ResultCode": 1200,
                    "Identifier": "Component1ID"
               }
          },
          {
               "AlertType": "urn:oma:at:scomo:1.0:OperationComplete",
               "SourceURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Download/Package1/Operations/DownloadInstall",
               "TargetURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Inventory/Deployed/Component2",
               "Mark": "warning",
               "DataType": "application/json",
               "Data": {
                    "ResultCode": 1200,
                    "Identifier": "Component2ID"
               }
          }
     ]
}
```

In the other hand, DM 1.x Generic Alert data can be directly delivered in the DM 2.0. For this, the Data JSON object has the name/value pair where the name is set to "DM1x", and the value is copied from the <Data> element in the DM 1.x. In case that the CDATA is used in the <Data> element, the CDATA MUST be eliminated. DM 2.0 Generic Alert using this approach is as follows:

```
{
     "Alert": [
          {
               "AlertType": "urn:oma:at:scomo:1.0:OperationComplete",
               "SourceURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Download/Package1/Operations/DownloadInstall",
               "TargetURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Inventory/Deployed/Component1",
               "Mark": "warning",
               "DataType": "text/xml",
               "Data": {
                    "DM1x": "<ResultCode>1200</ResultCode>          <!-- SCOMO
Result Code --><Identifier>Component1ID</Identifier>"
               }
          },
          {
               "AlertType": "urn:oma:at:scomo:1.0:OperationComplete",
```

```
                "SourceURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Download/Package1/Operations/DownloadInstall",
                "TargetURI": "urn:oma:mo:oma-
scomo:1.0/SCOMO1/Inventory/Deployed/Component2",
                "Mark": "warning",
                "DataType": "text/xml",
                "Data": {
                    "DM1x": "<ResultCode>1200</ResultCode>          <!-- SCOMO
Result Code --><Identifier>Component1ID</Identifier>"
                }
            }
        ]
}
```

It is up to the implementation to use which approaches, and other approaches are not precluded as well.

13.2 DM 1.x ACL Mechanism Interworking

For the access control, DM 1.x defines the ACL mechanism that allows the access control at the node resolution, which means that each node can have different ACL configurations. DM 2.0 simplifies the access control mechanism by introducing the access control at the MO instance resolution. However, Management Object may require the access control at the node resolution for the proper operations depending on the MO design. Since DM 2.0 intends to work with every Management Object designed for DM 1.x, DM 2.0 provides an optional feature that provide the access control at the node resolution. The optional *<MOID>/<MIID>/<x>/Path* node in the Delegation Access Control MO can provide the access control at the node resolution as specified in the section 12.2.1.

The DM Client SHOULD support the *<MOID>/<MIID>/<x>/Path* node when the DM Client needs to support MO instance that requires the access control at the node level.

13.3 DM 1.x DDF Interworking

OMA DM 2.0 uses the DM 1.3 DDF [DM_1.3], but unnecessary elements in DM 1.3 DDF might be silently ignored. The supported DDF elements are specified in the section 6.4.

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
OMA-TS-DM_Protocol-V2_0-20160209-A	09 Feb 2016	Status changed to Approved by TP TP Ref # OMA-TP-2016-0032-INP_DM_V2_0_ERP_for_final_Approval

Appendix B. Static Conformance Requirements (Normative)

The notation used in this appendix is specified in [SCRRULES].

B.1 SCR for DM Client

B.1.1 DM Packages

Item	Function	Reference	Requirement
DM-PACK-001-C-M	Support of DM Package flow	Section 5.2	
DM-PACK-002-C-O	Support of Package#0	Section 5.2.1	DM-PACK-003-C-O AND DM-PACK-004-C-O AND (DM-PACK-005-C-O OR DM-PACK-006-C-O OR DM-PACK-007-C-O OR DM-PACK-008-C-O OR DM-PACK-009-C-O OR DM-PACK-010-C-O)
DM-PACK-003-C-O	Support of VER header	Section 5.2.1.1	
DM-PACK-004-C-O	Support of OC header	Section 5.2.1.1	
DM-PACK-005-C-O	Support of SERVER-ID option	Section 5.2.1.2	
DM-PACK-006-C-O	Support of PREFERRED-CON-TYPE option	Section 5.2.1.2	
DM-PACK-007-C-O	Support of NOTIFICATION-ID option	Section 5.2.1.2	
DM-PACK-008-C-O	Support of SHA256-DIGEST option	Section 5.2.1.2	
DM-PACK-009-C-O	Support of TIMESTAMP option	Section 5.2.1.2	
DM-PACK-010-C-O	Support of REQ-MOS option	Section 5.2.1.2	
DM-PACK-011-C-M	Support of Package#1	Section 5.2.2	
DM-PACK-012-C-M	Support of Package#2	Section 5.2.3	
DM-PACK-013-C-M	Support of Package#3	Section 5.2.4	
DM-PACK-014-C-M	Support of HGET Command	Section 5.3	
DM-PACK-015-C-M	Support of HPUT Command	Section 5.3	
DM-PACK-016-C-M	Support of HPOST Command	Section 5.3	
DM-PACK-017-C-M	Support of DELETE Command	Section 5.3	
DM-PACK-018-C-M	Support of EXEC Command	Section 5.3	
DM-PACK-019-C-O	Support of GET Command	Section 5.3	
DM-PACK-020-C-O	Support of SHOW Command	Section 5.3	
DM-PACK-021-C-M	Support of CONT Command	Section 5.3	
DM-PACK-022-C-M	Support of END	Section 5.3	

Item	Function	Reference	Requirement
	Command		
DM-PACK-023-C-O	Support of DEFAULT Command	Section 5.3	
DM-PACK-024-C-O	Support of SUB Command	Section 5.3	DM-PACK-025-C-O
DM-PACK-025-C-O	Support of UNSUB Command	Section 5.3	
DM-PACK-026-C-M	Support of Generic Alert Mechanism	Section 5.4	
DM-PACK-027-C-M	Support of Synchronous reporting mechanism	Section 5.4.1	
DM-PACK-028-C-M	Support of Asynchronous reporting mechanism	Section 5.4.1	

B.1.2 DM Object

Item	Function	Reference	Requirement
DM-OBJ-001-C-M	Support of ClientURI	Section 6.1	
DM-OBJ-002-C-O	Support of query component, the x-name component and the wildcard of ClientURI	Section 6.1	
DM-OBJ-003-C-M	Support of ServerURI	Section 6.2	
DM-OBJ-004-C-M	Support of DevID ServerURI value	Section 6.2	
DM-OBJ-005-C-M	Support of MNC ServerURI value	Section 6.2	
DM-OBJ-006-C-M	Support of MCC ServerURI value	Section 6.2	
DM-OBJ-007-C-M	Support of CMNC ServerURI value	Section 6.2	
DM-OBJ-008-C-M	Support of CMCC ServerURI value	Section 6.2	
DM-OBJ-009-C-M	Support of SPName ServerURI value	Section 6.2	
DM-OBJ-010-C-M	Support of MO definition	Section 6.3	
DM-OBJ-011-C-O	Support of any MO compliant to DM definition	Section 6.3	
DM-OBJ-012-C-M	Allowing the execution of a DM Command only if DM Server has access rights on all the nodes	Section 6.3.1.1	
DM-OBJ-013-C-M	Support of MgmtTree DDF Element	Section 6.4	
DM-OBJ-014-C-M	Support of VerDTD DDF Element	Section 6.4	
DM-OBJ-015-C-O	Support of Man DDF Element	Section 6.4	
DM-OBJ-016-C-O	Support of Mod DDF	Section 6.4	

Item	Function	Reference	Requirement
	Element		
DM-OBJ-017-C-M	Support of Node DDF Element	Section 6.4	
DM-OBJ-018-C-M	Support of NodeName DDF Element	Section 6.4	
DM-OBJ-019-C-O	Support of Path DDF Element	Section 6.4	
DM-OBJ-020-C-O	Support of Value DDF Element	Section 6.4	
DM-OBJ-021-C-M	Support of b64, bin, bool, chr, int, node, null, xml, date, time, float DDF elements	Section 6.4	
DM-OBJ-022-C-O	Support of MIME DDF Element	Section 6.4	
DM-OBJ-023-C-M	Support of DDFName DDF Element	Section 6.4	
DM-OBJ-024-C-M	Support of DFProperties DDF Element	Section 6.4	
DM-OBJ-025-C-M	Support of AccessType DDF Element	Section 6.4	
DM-OBJ-026-C-O	Support of DefaultValue DDF Element	Section 6.4	
DM-OBJ-027-C-O	Support of Description DDF Element	Section 6.4	
DM-OBJ-028-C-O	Support of DFFormat DDF Element	Section 6.4	
DM-OBJ-029-C-O	Support of Occurrence DDF Element	Section 6.4	
DM-OBJ-030-C-O	Support of Scope DDF Element	Section 6.4	
DM-OBJ-031-C-O	Support of DFTitle DDF Element	Section 6.4	
DM-OBJ-032-C-O	Support of DFType DDF Element	Section 6.4	
DM-OBJ-033-C-O	Support of CaseSense DDF Element	Section 6.4	

B.1.3 Package Serialization

Item	Function	Reference	Requirement
DM-SER-001-C-M	Support of Package#0 Binary format	Section 7.1.1	
DM-SER-002-C-O	Support of Package#0 other format	Section 7.1.2	
DM-SER-003-C-M	Support of JSON format for Package#1,Package#2 and Package#3, MO, Generic Alert	Section 7.2.1	
DM-SER-004-C-O	Support of other serialization format	Section 7.2.2	

B.1.4 Bootstrap

Item	Function	Reference	Requirement
DM-BOOT-001-C-M	Support of at least one bootstrap mechanism	Section 8	
DM-BOOT-002-C-O	Support of Smartcard Bootstrap	Section 8.1	DM-BOOT-010-C-O AND DM-BOOT-012-C-O
DM-BOOT-003-C-O	Support of Factory Bootstrap	Section 8.2	
DM-BOOT-004-C-O	Support of Client Initiated Bootstrap	Section 8.3	
DM-BOOT-005-C-O	Support of Server Initiated Bootstrap	Section 8.4	
DM-BOOT-006-C-O	Support of Smartcard Bootstrap with Secure Channel	Section 8.1 Appendix G	DM-BOOT-011-C-O AND DM-SEC-013-C-0 AND DM-BOOT-012-C-O
DM-BOOT-007-C-M	Support of Bootstrap Security	Section 9.3	
DM-BOOT-008-C-O	Device supports a Smartcard	Section 8.1	DM-BOOT-002-C-O OR DM-BOOT-006-C-O
DM-BOOT-009-C-O	Smartcard Bootstrap function is enabled by DM client and the smartcard has not been rejected by the device	Section 8.1	
DM-BOOT-010-C-O	Retrieve Bootstrap Data from Smartcard, process and apply it to the Device configuration when Bootstrap from Smartcard is not disabled by Client and not rejected by Device	Section 8.1	
DM-BOOT-011-C-O	Retrieve Bootstrap Data from Smartcard using Secure Channel, process and apply it to the Device configuration when Bootstrap from Smartcard is not disabled by Client and not rejected by Device	Section 8.1	
DM-BOOT-012-C-O	DM Client removes the DM Server's information from the Device Management Tree if they are no longer stored on the Smartcard when the device is switched on	Section 8.1	

B.1.5 Security

Item	Function	Reference	Requirement
DM-SEC-001-C-M	Support for Basic Authentication Schema	Section 9.2.1	
DM-SEC-002- C-O	Digest Authentication Schema	Section 9.2.1	
DM-SEC-003- C-M	X.509 certificate	Section 9.2.1	
DM-SEC-004- C-M	Support of at least one mode: X509 certificate and PSK-TLS	Section 9.2.2	
DM-SEC-005- C-O	Support for TLS1.0	Section 9.2.2	
DM-SEC-006- C-M	Support of TLS1.1	Section 9.2.2	
DM-SEC-007- C-O	Support of TLS1.2	Section 9.2.2	
DM-SEC-008- C-O	Support of PSK-TLS mode	Section 9.2.2	DM-SEC-009- C-O
DM-SEC-009- C-O	Support of at least one Cipher Suite, when using PSK-TLS: TLS_PSK_WITH_AES_128_GCM_SHA256 , TLS_DHE_PSK_WITH_AES_128_GCM_SHA256, TLS_RSA_PSK_WITH_AES_128_GCM_SHA256, TLS_PSK_WITH_AES_128_CBC_SHA256, TLS_DHE_PSK_WITH_AES_128_CBC_SHA256, TLS_RSA_PSK_WITH_AES_128_CBC_SHA256 TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256	Section 9.2.2	
DM-SEC-010- C-O	Support of X.509 mode	Section 9.2.2	DM-SEC-011- C-O

DM-SEC-011- C-O	Support of at least one Cipher Suite, when using X.509 mode: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, TLS_RSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, TLS_RSA_WITH_AES_128_CBC_SHA256	Section 9.2.2	
DM-SEC-012- C-M	Check that the Server is using TLS1.0 or TLS1.1 or TLS1.2	Section 9.2.2	
DM-SEC-013-C-0	Support of Smartcard Secure Channel	Section 8	DM-SEC-014-C-O
DM-SEC-014-C-O	Smartcard Secure Channel using [GLOBALPLATFORM] [GP SCP03]	Section 8.1 Appendix G	

B.1.6 Mechanisms

Item	Function	Reference	Requirement
DM-MEC-001-C-O	Support of Access Control Mechanism	Section 10	DM-STDOBJ-003-C-O
DM-MEC-002-C-O	Support of MO Cache Mechanism	Section 11	
DM-MEC-003-C-O	Support of at least one notification mechanism	Appendix E	DM-MEC-004-C-O OR DM-MEC-005-C-O OR DM-MEC-006-C-O
DM-MEC-004-C-O	Support of Connectionless WAP Push mechanism	Appendix E.1	
DM-MEC-005-C-O	Support of Google Cloud Messaging notification mechanism	Appendix E.2	
DM-MEC-006-C-O	Support of other notification mechanism	Appendix E.3	

B.1.7 Standard Object

Item	Function	Reference	Requirement
DM-STDOBJ-001-C-M	Support of DevInfo MO	Section 12.1	
DM-STDOBJ-002-C-M	Support of DM Account MO	Section 12.2	
DM-STDOBJ-003-C-O	Support of Delegation Access Control MO	Section 12.2.1	
DM-STDOBJ-004-C-M	Support of Session Information MO	Section 12.4	

B.2 SCR for DM Server

B.2.1 DM Packages

Item	Function	Reference	Requirement
DM-PACK-001-S-M	Support of DM Package flow	Section 5.2	
DM-PACK-002-S-M	Support of Package#0	Section 5.2.1	
DM-PACK-003-S-M	Support of VER header	Section 5.2.1.1	
DM-PACK-004-S-M	Support of OC header	Section 5.2.1.1	
DM-PACK-005-S-M	Support of SERVER-ID option	Section 5.2.1.2	
DM-PACK-006-S-M	Support of PREFERRED-CON-TYPE option	Section 5.2.1.2	
DM-PACK-007-S-M	Support of NOTIFICATION-ID option	Section 5.2.1.2	
DM-PACK-008-S-M	Support of SHA256-DIGEST option	Section 5.2.1.2	
DM-PACK-009-S-M	Support of TIMESTAMP option	Section 5.2.1.2	
DM-PACK-010-S-M	Support of REQ-MOS option	Section 5.2.1.2	
DM-PACK-011-S-M	Support of Package#1	Section 5.2.2	
DM-PACK-012-S-M	Support of Package#2	Section 5.2.3	
DM-PACK-013-S-M	Support of Package#3	Section 5.2.4	
DM-PACK-014-S-M	Support of HGET Command	Section 5.3	
DM-PACK-015-S-M	Support of HPUT Command	Section 5.3	
DM-PACK-016-S-M	Support of HPOST Command	Section 5.3	
DM-PACK-017-S-M	Support of DELETE Command	Section 5.3	
DM-PACK-018-S-M	Support of EXEC Command	Section 5.3	
DM-PACK-019-S-M	Support of GET Command	Section 5.3	
DM-PACK-020-S-M	Support of SHOW Command	Section 5.3	

Item	Function	Reference	Requirement
DM-PACK-021-S-M	Support of CONT Command	Section 5.3	
DM-PACK-022-S-M	Support of END Command	Section 5.3	
DM-PACK-023-S-M	Support of DEFAULT Command	Section 5.3	
DM-PACK-024-S-O	Support of SUB Command	Section 5.3	DM-PACK-025-S-O
DM-PACK-025-S-O	Support of UNSUB Command	Section 5.3	
DM-PACK-026-S-M	Support of Generic Alert Mechanism	Section 5.4	
DM-PACK-027-S-M	Support of Synchronous reporting mechanism	Section 5.4.1	
DM-PACK-028-S-M	Support of Asynchronous reporting mechanism	Section 5.4.1	
DM-PACK-029-S-O	Support of *OMADM-Accept* HTTP header	Section 5.5	

B.2.2 DM Object

Item	Function	Reference	Requirement
DM-OBJ-001-S-M	Support of ClientURI	Section 6.1	
DM-OBJ-002-S-M	Support of query component, the x-name component and the wildcard of ClientURI	Section 6.1	
DM-OBJ-003-S-M	Support of ServerURI	Section 6.2	
DM-OBJ-004-S-M	Support of DevID ServerURI value	Section 6.2	
DM-OBJ-005-S-M	Support of MNC ServerURI value	Section 6.2	
DM-OBJ-006-S-M	Support of MCC ServerURI value	Section 6.2	
DM-OBJ-007-S-M	Support of CMNC ServerURI value	Section 6.2	
DM-OBJ-008-S-M	Support of CMCC ServerURI value	Section 6.2	
DM-OBJ-009-S-M	Support of SPName ServerURI value	Section 6.2	
DM-OBJ-010-S-M	Support of MO definition	Section 6.3	
DM-OBJ-011-S-O	Support of any MO compliant to DM definition	Section 6.3	
DM-OBJ-012-S-M	Support of MgmtTree DDF Element	Section 6.4	
DM-OBJ-013-S-M	Support of VerDTD DDF Element	Section 6.4	
DM-OBJ-014-S-O	Support of Man DDF Element	Section 6.4	

Item	Function	Reference	Requirement
DM-OBJ-015-S-O	Support of Mod DDF Element	Section 6.4	
DM-OBJ-016-S-M	Support of Node DDF Element	Section 6.4	
DM-OBJ-017-S-M	Support of NodeName DDF Element	Section 6.4	
DM-OBJ-018-S-O	Support of Path DDF Element	Section 6.4	
DM-OBJ-019-S-O	Support of Value DDF Element	Section 6.4	
DM-OBJ-020-S-M	Support of b64, bin, bool, chr, int, node, null, xml, date, time, float DDF elements	Section 6.4	
DM-OBJ-021-S-O	Support of MIME DDF Element	Section 6.4	
DM-OBJ-022-S-M	Support of DDFName DDF Element	Section 6.4	
DM-OBJ-023-S-M	Support of DFProperties DDF Element	Section 6.4	
DM-OBJ-024-S-M	Support of AccessType DDF Element	Section 6.4	
DM-OBJ-025-S-O	Support of DefaultValue DDF Element	Section 6.4	
DM-OBJ-026-S-O	Support of Description DDF Element	Section 6.4	
DM-OBJ-027-S-O	Support of DFFormat DDF Element	Section 6.4	
DM-OBJ-028-S-O	Support of Occurrence DDF Element	Section 6.4	
DM-OBJ-029-S-O	Support of Scope DDF Element	Section 6.4	
DM-OBJ-030-S-O	Support of DFTitle DDF Element	Section 6.4	
DM-OBJ-031-S-O	Support of DFType DDF Element	Section 6.4	
DM-OBJ-032-S-O	Support of CaseSense DDF Element	Section 6.4	

B.2.3 Package Serialization

Item	Function	Reference	Requirement
DM-SER-001-S-M	Support of Package#0 Binary format	Section 7.1.1	
DM-SER-002-S-O	Support of Package#0 other format	Section 7.1.2	
DM-SER-003-S-M	Support of JSON format for Package#1,Package#2 and Package#3, MO, Generic Alert	Section 7.2.1	
DM-SER-004-S-O	Support of other serialization format	Section 7.2.2	DM-PACK-029-S-O

B.2.4 Bootstrap

Item	Function	Reference	Requirement
DM-BOOT-001-S-M	Support of Server Initiated Bootstrap	Section 8.4	

B.2.5 Security

Item	Function	Reference	Requirement
DM-SEC-001-S-M	Support for Basic Authentication Schema	Section 9.2.1	
DM-SEC-002- S-M	Digest Authentication Schema	Section 9.2.1	
DM-SEC-003- S-M	X.509 certificate	Section 9.2.1	
DM-SEC-004- S-M	Support of X.509 Certificate Mode and PSK-TLS mode	Section 9.2.2	
DM-SEC-005- S-O	Support for TLS1.0	Section 9.2.2	
DM-SEC-006- S-M	Support for TLS1.1	Section 9.2.2	
DM-SEC-007- S-O	Support for TLS1.2	Section 9.2.2	
DM-SEC-008- S-M	Support for PSK-TLS mode	Section 9.2.2	

DM-SEC-009- S-M	Support of these Cipher Suites when using PSK-TLS mode: TLS_PSK_WITH_AES_128_GCM_SHA256 , TLS_DHE_PSK_WITH_AES_128_GCM_SHA256, TLS_RSA_PSK_WITH_AES_128_GCM_SHA256, TLS_PSK_WITH_AES_128_CBC_SHA256, TLS_DHE_PSK_WITH_AES_128_CBC_SHA256, TLS_RSA_PSK_WITH_AES_128_CBC_SHA256, TLS_ECDHE_PSK_WITH_AES_128_CBC_SHA256	Section 9.2.2	
DM-SEC-010- S-M	Support of X.509 mode	Section 9.2.2	
DM-SEC-011- S-M	Support of these Cipher Suites when using X.509 mode: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, TLS_DHE_RSA_WITH_AES_128_GCM_SHA256, TLS_RSA_WITH_AES_128_GCM_SHA256, TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256, TLS_DHE_RSA_WITH_AES_128_CBC_SHA256, TLS_RSA_WITH_AES_128_CBC_SHA256	Section 9.2.2	

B.2.6 Mechanisms

Item	Function	Reference	Requirement
DM-MEC-001-S-M	Support of Access Control Mechanism	Section 10	
DM-MEC-002-S-O	Support of MO Cache Mechanism	Section 11	
DM-MEC-004-S-M	Support of Connectionless WAP Push mechanism	Appendix E.1	
DM-MEC-004-S-O	Support of Google Cloud Messaging notification mechanism	Appendix E.2	
DM-MEC-005-C-O	Support of other notification mechanism	Appendix E.3	

B.2.7 Standard Object

Item	Function	Reference	Requirement
DM-STDOBJ-001-S-M	Support of DevInfo MO	Section 12.1	
DM-STDOBJ-002-S-M	Support of DM Account MO	Section 12.2	
DM-STDOBJ-003-S-M	Support of Delegation Access Control MO	Section 12.2.1	
DM-STDOBJ-004-S-M	Support of Session Information MO	Section 12.4	

Appendix C. Response Status Codes for DM Commands (Normative)

The response status codes for DM commands are a numeric value. The codes are divided into 4 classes. The only valid values are the standard values defined in this specification.

Status Codes	Label	Reason Phrase	Applied DM Commands
Successful 2xx			
200	OK	The DM command completed successfully: • HPUT/HPOST: at least one of the requested ClientURI is successfully sent to the Data Repository • HGET: the retrieved data is successfully stored at the device • SHOW: the UI session is successfully initiated regardless of whether the user interaction is successful or not • DEFAULT: the command is successfully received	All
202	Accepted	Accepted for processing. The asynchronous reporting mechanism is used to report the actual results.	EXEC
204	No Content	The request was successfully completed but no data is being returned. The response code is also returned in response to the GET command when the target has no content.	GET
Redirection 3xx			
304	Not Modified	The request carries the cache validator and the access is allowed, but the MO data identified by the ClientURI is not modified according to the cache validation process.	GET, HPUT, HPOST
Originator Exceptions 4xx			
400	Bad Request	The requested command could not be performed because of malformed syntax in the command.	All
403	Forbidden	The requested command failed because the sender does not have adequate access rights on the recipient.	Except SHOW, CONT, END
404	Not Found	The requested target was not found: • HPUT/HPOST: all specified ClientURIs were not found • HGET, DELETE, EXEC, GET, DEFAULT, SUB: the specified ClientURI was not found • UNSUB: the specified ClientURI was not used in a previous SUB Command	Except SHOW, CONT, END
405	Command Not Allowed	The requested command is not allowed on the node identified by the ClientURI: • EXEC: ClientURI identifies the non-executable node • DELETE: the node identified by the ClientURI is mandatory	EXEC, DELETE
406	Not Acceptable	The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request: • HPUT, HPOST: data formats requested by DM Server are not supported by DM Client	

415	Unsupported Media Type	The request is refused because the request uses a format not supported by the requested resource for the requested method: • HGET: the retrieved data by the HTTP GET is unrecognized	HGET
419	ServerURI Error	The ServerURI provided causes errors: • HGET: HTTP GET to the ServerURI is failed, and no data is received • HPUT/HPOST: at least one of the ClientURIs is ready to be sent to the ServerURI. But, HTTP PUT/POST request to the ServerURI is failed	HGET, HPUT, HPOST
Recipient Exception 5xx			
500	Internal Error	The recipient encountered an unexpected condition which prevented it from fulfilling the request.	All
501	Not Implemented	The recipient does not support the features to fulfil the request. This is the appropriate response when the recipient does not recognize the requested command and is not capable of supporting it for any resource.	All
503	Service Unavailable	The recipient is currently unable to handle the request due to a temporary overloading or maintenance of the recipient. The implication is that this is a temporary condition; which will be alleviated after some delay.	All
506	Device Full	The response indicates that the recipient has not enough storage space for the data.	HGET
507	User Rejected	The request is not executed since the user rejected the request.	All
Application specific codes 1xxx			
1000 – 1999		These status codes are application specific status codes and the meanings of these are not defined in this specification. It is recommended to define status codes with the same grouping as above within this application specific interval but it is the application that defines the allowed values: Successful 12xx Redirection 13xx Originator Exceptions 14xx Recipient Exception 15xx	Application Dependent

Appendix D. Alert Type Table (Normative)

The following table specifies the list of AlertType values which refer to OMA DM 2.0 Protocol operations.

MO specifications can defined other AlertType values.

AlertType	Description	Data Content
urn:oma:at:dm:2.0:ServerInitiatedMgmt	Specifies a server-initiated DM Session.	NOTIFICATION-ID (see 5.2.1)
urn:oma:at:dm:2.0:ClientInitiatedMgmt	Specifies a client-initiated DM Session.	
urn:oma:at:dm:2.0:BootstrapComplete	Specifies a client-initiated DM Session after successful bootstrap	

Appendix E. DM Notification Delivery and Transport (Normative)

DM Notification can be delivered from the DM Server to the DM Client using various transports. In this specification some transport bindings area providedOther transports are not precluded and can be used on their availabilities.

If Device supports any of the described transports, the DM Client MUST support at least one of specified Notification Delivery mechanism.

E.1 Connectionless WAP Push

The DM Notification can be sent to the DM Client using the Push OTA Protocol over WSP (OTA-WSP) [PUSHOTA] with the following additional rules:

- The package MUST be sent using the non-secure connectionless push.
- Application-ID *0x07* MUST be used.
- Content-Type Code *0x58* MUST be used (*application/vnd.syncml.dm.notification*).
- Other Push header fields may be included; however the total length of the Push header MUST NOT exceed 48 bytes (to ensure that there is sufficient space for the Push message body that contains the DM Notification).

The DM Server MAY support Connectionless WAP Push mechanism. The DM Client MAY support Connectionless WAP Push mechanism.

For devices on cellular networks, connectionless WAP Push is typically delivered over SMS. For IP-capable devices, connectionless WAP Push can be delivered over UDP. In order to receive non-secure connectionless WAP Push over UDP, an IP-capable device MUST listen to the IANA registered port number for connectionless WAP Push (i.e. 2948).

E.1.1 Using non WAP Push capable devices

If the receiver is not a WAP device, it is very unlikely that any other application would be active on the same port, which has been publicly registered with IANA. The decoding of the message headers is very straightforward even if the device lacks a full WAP stack and therefore the device MUST examine if the message has been sent to the default WAP push port (2948) and if the Application-ID and the Media Type are one assigned to the OMA DM Notification Initiation Package. If this information is correct then the message MUST be routed to the OMA Device Management application.

E.2 Google Cloud Messaging

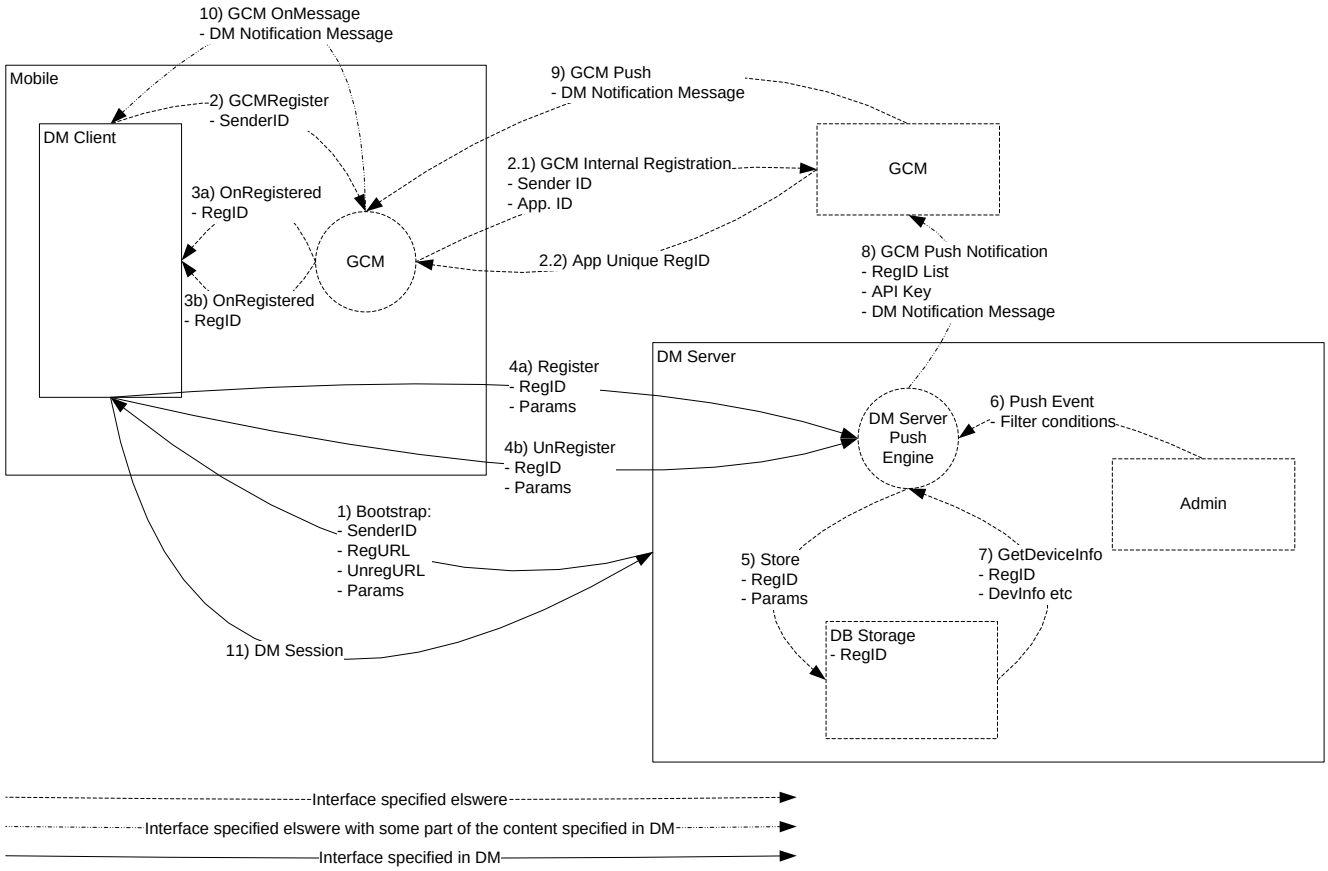
Google Cloud Messaging for Android [GCM] is a service that allows the 3rd party application server to send data to Android applications running on Android based device. Using GCM, the DM Notification can be delivered from the DM Server to the DM Client: in this case, the DM Server takes the role of the 3rd party application server, and the DM Client runs as an Android application in the device.

DM Servers MAY support GCM. DM Client MAY support GCM.

The following chapters specify the usage of GCM.

E.2.1 GCM Overview

This is an example message flow based on GCM architecture:



E.2.2 Message Flow

The procedure to support GCM is:

1 The DM Server MUST deliver the DM Server specific GCM configuration as part of the DM Bootstrap

2 The DM Client MUST register itself to the GCM service

3-5 When the DM Client receives the OnRegister or UnRegister events from the GCM service, it MUST respectively register or unregister itself to the DM Server. This registration MUST contain the RegID which is the GCM identifier for the specific DM Client instance and which the DM Server MUST store

6-10 When the DM Server wants to initiate a DM Session to a specific DM Client it MUST send a DM Notification via the GCM interface using the previous stored identified

11 The DM Client MUST handle the DM Notification received

E.2.3 Bootstrap (Interface 1)

The DM Server MAY indicate for the DM Client that the DM Servers supports GCM to send DM Notification populating the related nodes in DM Account MO as part of the DM Bootstrap. Registration and Unregistration (Interface 4)

The DM Client MUST send an HTTP POST to the *Push/GCM/RegURL* or to the *Push/GCM/UnREgURL* that is provisioned during the bootstrap for registration or unregistration respectively. The Media Type MUST be “application/x-www-form-urlencoded;charset=UTF-8”. The HTTP body MUST contain the key “RegId” with the value of the RegId that was received

from the GCM Service with the value of the *Push/GCM/RegParams* node if that was included in the bootstrap. This string MUST be encoded as a query string as defined in the URI. The same security specified for DM-2 MUST implemented.

E.2.4 Push Notification (Interface 8,10)

The content of the DM Notification MUST be encoded as base64 string. The DM Server MUST send the Push Message to the GCM Service including the key “SMS” with the value of the base64 encoded DM Notification. DM Client MUST process the push message once received from the GCM Service.

Informative Note: other encoding MAY be used according to GCM Service specification.

E.3 Other Transport Mechanisms

DM Server and DM Client MAY use other transport mechanism to convey DM Notification.

Appendix F. Using Multipart Content-Type for Response (Normative)

When the DM Server sends the GET command to the DM Client, the DM Client is requested to return the requested MO data. Those data is not included in the Package#3, but they are embedded in the HTTP message body (i.e., the HTTP POST message) that carries the Package#3.

If there is no data to be embedded (e.g., the status codes for all GET commands are either "404 Not Found" or "304 Not Modified"), then the *Content-Type* HTTP header defined for Package#3 MUST be used (see chapter 7).

If data is available, the “multipart/form-data” *Content-Type* HTTP header MUST be used. The first encapsulation of the multipart-body MUST be the Package#3 as specified in the section 7, then an encapsulation MUST be provided for data corresponding to each GET command sent by DM Server with status code "200 OK". Every encapsulation MUST be identified by the name attribute specified in the *Content-Disposition* HTTP header; the sequence order of the status code starts from 0. Multipart, encapsulation, multipart-body and name are defined in [RFC1521].

If the ClientURI is resolved into a single leaf node, the value of the leaf node MAY be directly embedded into the encapsulation specifying the Media Type of the node value as the *Content-Type* HTTP header. Alternatively, MO serialization MAY be used in which case the related Media Type MUST be used as value (e.g., application/dmmo+json).

This is an example of the Package#2 including GET commands:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.oma.dm.request+json

{
     "CMD": [
          ["GET", "urn:oma:mo:oma-dm-devinfo:1.2//DevID"],
          ["GET", "urn:oma:mo:oma-dm-devinfo:1.2//DevID"],
          ["GET", "urn:oma:mo:oma-sessioninfomo:1.0//"],
          ["HPUT", "http://www.data.com/MOData", "urn:oma:mo:oma-moid:1.0//"],
          ["GET", "urn:oma:mo:invalid-moid:1.0//"],
          ["HGET", "http://www.dms1.com/","urn:oma:mo:oma-dm-
dmacc:2.0/dms1/AuthNoti/AuthSecret"]
     ]
}
```

This is an example of response using the “multipart/form-data” *Content-Type* HTTP header:

```
POST /dmclient/dm20 HTTP/1.1
Content-Type: multipart/form-data; boundary=simple_boundary
Accept: application/vnd.oma.dm.request+json
OMADM-DevID: IMEI:493005100592800
Host: www.devicemgmt.org

--simple_boundary
Content-Disposition: form-data
Content-Type: application/vnd.oma.dm.response+json

{
     "Status": [
          200,
          200,
          200,
          200,
          404,
          200
     ]
}
--simple_boundary
```

```
Content-Disposition: form-data; name="0"
Content-Type: text/plain

IMEI:493005100592800
--simple_boundary
Content-Disposition: form-data; name="1"
Content-Type: application/dmmo+json

{
     "DDF": "http://www.vendor.com/DDF/oma-dm-devinfo1.2.ddf",
     "ClientURI": "urn:oma:mo:oma-dm-devinfo:1.2//DevID",
     "MOData": {
          "DevID": "IMEI:493005100592800"
     }
}
--simple_boundary
Content-Disposition: form-data; name="2"
Content-Type: application/dmmo+json

{
     "DDF": "http://www.vendor.com/DDF/oma-sessioninfomo1.0.ddf",
     "ClientURI": "urn:oma:mo:oma-sessioninfomo:1.0//",
     "MOData": {
          "SessionInfoMO": {
               "CBT": 1,
               "ROAMING": 0
          }
     }
}
--simple_boundary--
```

Appendix G. Storage of DM Bootstrap Message on the Smartcard (Normative)

We can sort out three main types of smartcards used for wireless telecom networks, characterised by their physical and logical characteristics:

- SIM smartcards platforms [TS151.011]
- UICC smartcards platforms [TS102.221]
- R-UIM smartcards platform [C.S0023-B_v1.0]

This section aims at specifying the storage mechanism of DM Bootstrap on such Smartcard platform type.

For the purposes of this document the R-UIM is to be treated according to the rules defined for the SIM.

G.1 File structure

The information format is based on [PKCS#15] specification. The DM Bootstrap is located under the PKCS#15 directory allowing the card issuer to decide the identifiers and the file locations. The smartcard operations that are relevant include:

- Application selection
- Cardholder verification
- File access (select file, read, write)

The [PKCS#15] specification defines a set of files. Within the PKCS#15 application, the starting point to access these files is the Object Directory File (ODF). The EF (ODF) contains pointers to other directory files. These directory files contain information on different types of objects (authentication objects (PIN), data objects, etc). For the purpose of DM Bootstrap, EF (ODF) MUST contain the EF Record describing the DODF-bootstrap. The EF (ODF) is described in section D.4.1 and [PKCS#15].

EF (ODF) contains pointers to one or more Data Object Directory Files (DODF) in priority order (i.e. the first DODF has the highest priority). Each DODF is regarded as the directory of data objects known to the PKCS#15 application. For the purposes of DM bootstrapping, EF (DODF-bootstrap) contains pointer to the Bootstrap Message, namely DM Bootstrap File. The EF (DODF-bootstrap) is described in section D.4.2 and [PKCS#15].

The provisioning files are stored as PKCS#15 opaque data objects.

The support of smartcard Bootstrap Message will be indicated to the ME's user agent, by the presence in the EF DIR (see [TS102.221]) of an application template as defined here after.

The RECOMMENDED format of EF (DIR) is a linear fixed record in order to be in line with [TS102.221].

EF (DIR) MUST contain the application template used for a PKCS#15 application as defined in [PKCS#15]. Application template MUST consist of Application identifier (tag 0x4F) and Path (tag 0x51) information.

The EF (ODF) and EF (DODF-bootstrap) MUST be used by the ME to determine the path of the DM_Bootstrap file.

UICC smartcard platforms can support two modes of activation: 2G and 3G. UICC smartcard platform activated in a 2G mode has the logical characteristics of the SIM smartcard platform [TS151.011]. In that case, smartcard operations for accessing the Bootstrap Message conform to the ones defined for the SIM as specified in section D.2.

UICC smartcard platform activated in a 3G mode has the physical and logical characteristics according to [TS102.221]. In that case, smartcard operations for accessing the Bootstrap Message are specified in section D.3.

G.2 Bootstrap Message on SIM or UICC activated in 2G mode

G.2.1 Access to the file structure

To select the PKCS15 application, the Device MUST evaluate the PKCS#15 application template present in the EF (DIR), then the Device MUST use the indirect selection method as defined in [TS151.011] to select the application.

G.2.2 Files Overview

The file structure for the Bootstrap Message within the SIM smartcard is described below.

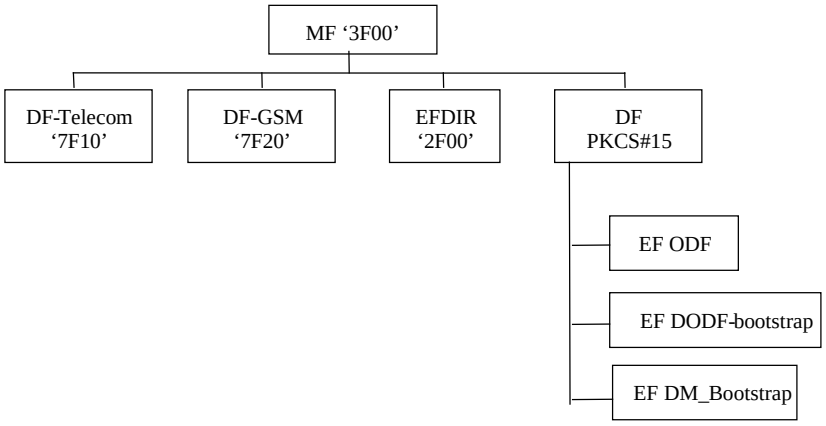


Figure 19: File structure for Bootstrap Message on SIM smartcard or 2G UICC

G.2.3 Access Method

SIM Commands Read Binary and Update Binary, as defined in [TS151.011], are used to access the Bootstrap Message.

G.2.4 Access Conditions

The Device is informed of the access conditions of the Bootstrap Message by evaluating the "private" and "modifiable" flags in the corresponding DODF-bootstrap files structure. When one of these flags is set cardholder verification is required. The CHV1 MUST be verified as defined in [TS151.011] when the "private" or "modifiable" flags are set.

Access conditions for files are proposed in the section D.4.

G.2.5 Requirements on the SIM or 2G UICC

To retrieve the Bootstrap Message from the SIM or 2G UICC, the Device MUST perform the following steps:

- Read EF (DIR) to evaluate the PKCS#15 application template and find the file identifier (and DF Path of the PKCS#15 DF),
- Select PKCS#15 DF (indirect selection), as defined in [TS151.011],
- Read ODF,
- Read DODF-bootstrap to locate the DM_Bootstrap file,
- Read the DM_Bootstrap file.

G.3 Bootstrap Message on UICC Activated in 3G Mode

G.3.1 Access to the file structure

To select the PKCS#15 application, the Device:

- MUST evaluate the PKCS#15 application template – i.e. PKCS#15 AID - present in the EF (DIR),
- MUST open a logical channel using UICC Command MANAGE CHANNEL as specified in [TS102.221],
- MUST select the PKCS#15 ADF using the PKCS#15 AID as parameter of the UICC Command SELECT, using direct application selection as defined in [TS102.221].

DM_Bootstrap file will be located under the PKCS#15 ADF.

G.3.2 Files Overview

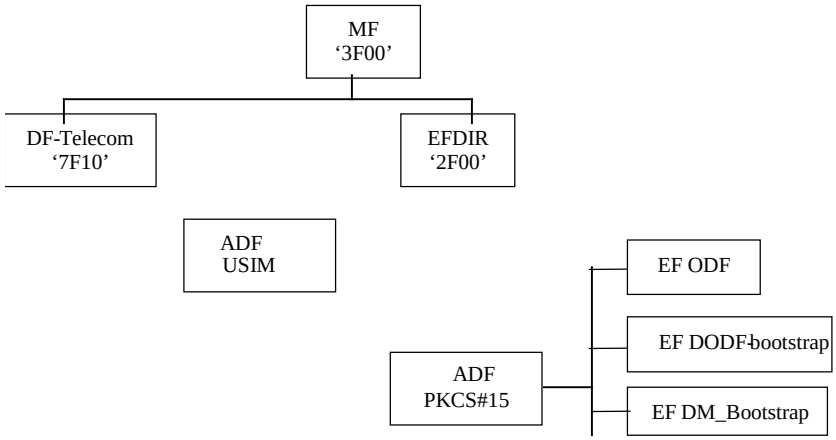


Figure 20: File structure for Bootstrap Message on 3G UICC

G.3.3 Access Method

UICC Commands Read Binary and Update Binary, as defined in [TS102.221], are used to access bootstrap data.

G.3.4 Access Conditions

The Device is informed of the access conditions of provisioning files by evaluating the "private" and "modifiable" flags in the corresponding DODF-bootstrap files structure.

In the case where one of the above mentioned flag is set, cardholder verification is required. The Device must evaluate the PIN references that must be verified as defined in [TS102.221] i.e. evaluate the FCP.

Access conditions for files are proposed in section D.4.

G.3.5 Requirements on the 3G UICC

To retrieve the Bootstrap Message from the 3G UICC, the Device MUST perform the following steps:

- Select PKCS#15 file structure as specified in G.3.1.
- Read ODF to locate the DODF-bootstrap,

- Read DODF-bootstrap to locate the DM_Bootstrap file,
- Read the DM_Bootstrap file

G.4 Files Description

All files defined are binary files as defined in [TS102.221]. These files are read and updated using SIM or UICSS Commands related to the application they belong to.

G.4.1 Object Directory File, EF ODF

The mandatory Object Directory File (ODF) ([PKCS#15], section 5.5.1) contains pointers to other EFs, each one containing a directory of PKCS#15 objects of a particular class (e.g. DODF-bootstrap). The File ID is specified in [PKCS#15]. The card issuer decides the file size. The EF (ODF) can be read but it MUST NOT be modifiable by the user.

In the case of SIM or UICC, the EF (ODF) is described below:

Identifier: default 0x5031, see [PKCS#15]	Structure: Binary	Mandatory
File size: decided by the card issuer	Update activity: low	
Access Conditions: READ ALW UPDATE ADM INVALIDATE ADM REHABILITATE ADM		
Description		
See [PKCS#15]		

G.4.2 Bootstrap Data Object Directory File, EF DODF-bootstrap

This Data Object Directory File provisioning contains directories of provisioning data objects ([PKCS#15], section 6.7) known to the PKCS#15 application.

The File ID is described in the EF (ODF). The file size depends on the number of provisioning objects stored in the smartcard. Thus, the card issuer decides the file size.

Identifier: 0x6420, See ODF	Structure: Binary	Mandatory
File size: decided by the card issuer	Update activity: low	
Access Conditions: READ ALW or CHV1 (SIM, See section D.2) or Universal / application / Local PIN (UICC, See section D.3) UPDATE ADM INVALIDATE ADM REHABILITATE ADM		
Description		
See hereafter and [PKCS#15]		

The EF (DODF-bootstrap) MUST contain information on provisioning objects:

- Readable label describing the provisioning document (CommonObjectAttributes.label). The ME could display this label to the user.
- Flags indicating whether the provisioning document is private (i.e., is protected with a PIN) and/or modifiable (CommonObjectAttributes.flags). The card issuer decides whether or not a file is private (it does not need to be if it does not contain any sensitive information)
- Object identifier indicating a DMboostrap object and the type of the provisioning object (CommonDataObjectAttributes.applicationOID)
- Pointer to the contents of the provisioning document (Path.path)

The EF (DODF-bootstrap) MUST contain the types of provisioning documents (indicated using object identifiers) to be used by the ME. The Bootstrap type is described hereafter.

A dedicated OID is required and defined for each provisioning file:

- Bootstrap OID = { joint-isu-itu-t(2) international-organizations(23) wap(43) oma-dm(7) dm-bootstrap(1)}

The ME MUST use the OID to distinguish the DODF-bootstrap from any other DODF. The EF (DODF-bootstrap) can be read but it MUST NOT be modifiable by the user.

G.4.3 EF DM_Bootstrap

Only the card issuer can modify EF DM_Bootstrap

Setting all bytes to 'FF' initialises EF DM_Bootstrap.

Identifier: See DODF	Structure: Binary	Optional
File size: decided by the card issuer	Update activity: low	
Access Conditions: READ ALW or CHV1 (SIM, See section D.2) or Universal / application / Local PIN (UICC, See section D.3) UPDATE ADM INVALIDATE ADM REHABILITATE ADM		
Description		
Contains a Bootstrap Message		

Appendix H. Secure channel between Smartcard and Device Storage for secure Bootstrap Data provisioning (Normative)

During DM Bootstrap procedure, sensitive data have to be provisioned in the Device. When Bootstrap information comes from Smartcard, a secure channel SHOULD be established. When required this secure channel SHALL follow the following procedure based on [GPSCP03] which is illustrated below. The Bootstrap information is retrieved from the Smartcard as described in Appendix G when the secure channel is established.

Pre-requisite: the Smartcard and the device have to share the same static Keys KEY_ENC, KEY_MAC, KEY_DEK as specified in [GPSCP03]. These keys are provisioned in the device using an out-of-band method.

The steps for the secure transfer are the following and are illustrated by the figure below (Figure 21):

- The PKSC#15 application used for transferring the Bootstrap information is selected
- Secure channel (mutual authentication) is established
- PKCS#15 flow as described in Appendix G takes place for selecting and transferring the Bootstrap file from Smartcard to the device: the sensitive Bootstrap data are transferred encrypted.

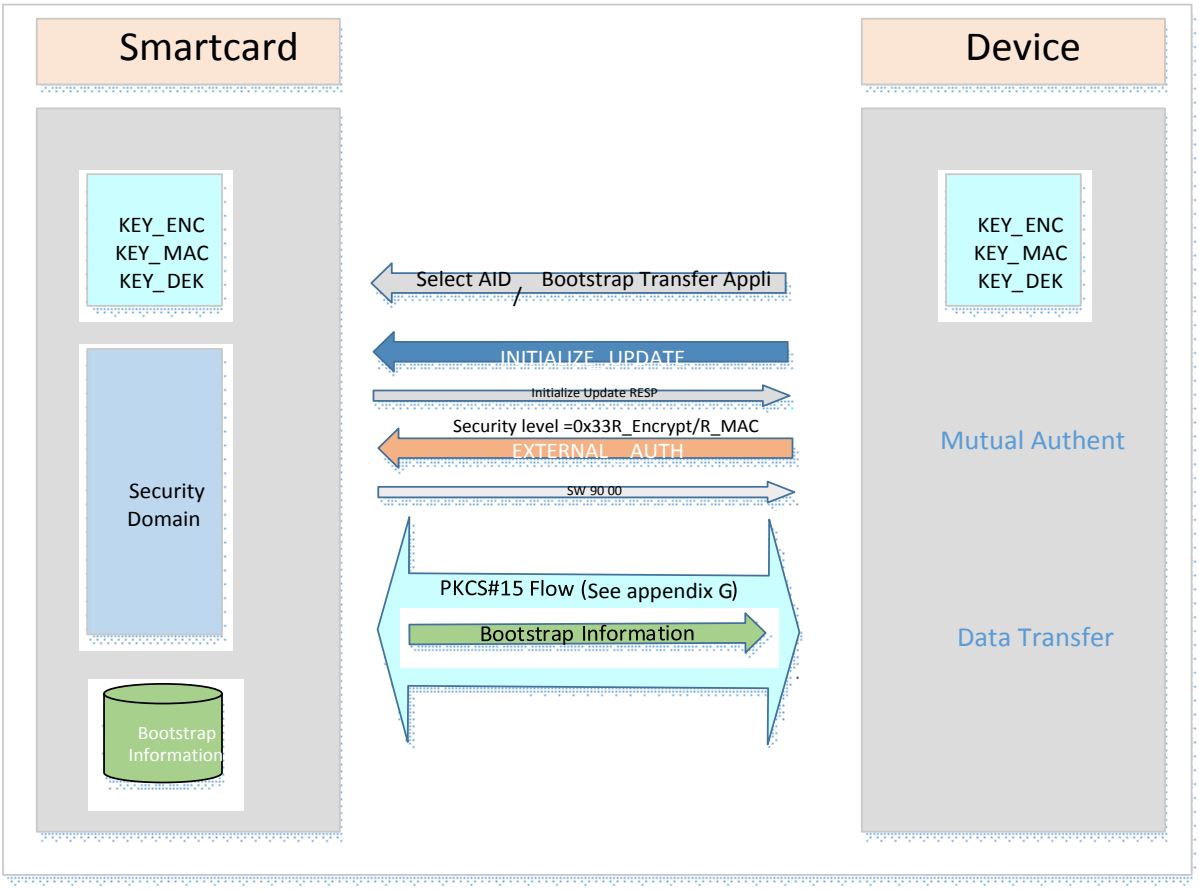


Figure 21: Bootstrap Information transfer from Smartcard to the DM Client using Secure Channel [GPSCP03]

Note 1: The INITIALIZE_UPDATE specifies the logical channel to use (CLA: 80H / 83H)

Note 2: The security level (P1) of the EXTERNAL_AUTH command is C-DECRYPTION, R-ENCRYPTION, C-MAC and R-MAC (P1=0x33)

Appendix I. Protocol Examples (Informative)

In this section several protocol scenarios will be demonstrated. In examples, mandatory HTTP header fields might be missing.

I.1 Examples for Retrieving MO data

In this section an example is presented in which the DM Server retrieves several MO data from the device using various DM commands (i.e., GET, HPOST and HPUT).

Package#1 that initializes the DM session is as follows (no Client Initiated Alerts and the DM Notification does not include the REQ-MOS Option):

```
POST /dmserver/dm20 HTTP/1.1
Content-Type: application/vnd.oma.dm.initiation+json
Accept: application/vnd.oma.dm.request+json
OMADM-DevID: IMEI:493005100592800
Host: www.dms.com

{
}
```

Package#2 to request the DevInfo Management Object and the list of all deployed software component identifiers is as follows:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.oma.dm.request+json

{
     "CMD": [
          ["GET", "oma:mo:oma-dm-devinfo:1.0//"],
          ["GET", "urn:oma:mo:oma-scomo:1.0//Inventory/Deployed/*/ID"]
     ]
}
```

For the GET command, multipart/form-data is used to send the Package#3 and to embed the requested MO data as specified in Appendix E. HTTP message for this is as follows:

```
POST /dmserver/dm20 HTTP/1.1
Content-Type: multipart/form-data; boundary=boundary
Accept: application/vnd.oma.dm.request+json
OMADM-DevID: IMEI:493005100592800
Host: www.dms.com

--boundary
Content-Disposition: form-data
Content-Type: application/vnd.oma.dm.response+json

{
     "Status": [
          200,
          200
     ]
}
--simple_boundary
Content-Disposition: form-data; name="0"
Content-Type: application/dmmo+json

[
     {
```

```
          "DDF": "http://www.vendor.com/DDF/devinfo1.0.ddf",
          "ClientURI": "oma:mo:oma-dm-devinfo:1.0//",
          "MOData": {
               "DevInfo": {
                    "DevID": "IMEI:493005100592800",
                    "Man": "Vendor",
                    "Mod": "DM_Client",
                    "DmV": "2.0",
                    "Lang": "en",
                    "DevType": "smartphone",
                    "OEM": "",
                    "FwV": "android4.0.4",
                    "SwV": "Vendor1.2",
                    "HwV": ""
               }
          }
     }
]
--simple_boundary
Content-Disposition: form-data; name="1"
Content-Type: application/dmmo+json

{
     {
          "DDF": "http://www.vendor.com/DDF/oma-scomo1.0.ddf",
          "ClientURI": "urn:oma:mo:oma-scomo:1.0//Inventory/Deployed/pkg1/ID",
          "MOData": {
               "ID": "pkg1_id"
          }
     },
     {
          "DDF": "http://www.vendor.com/DDF/oma-scomo1.0.ddf",
          "ClientURI": "urn:oma:mo:oma-scomo:1.0//Inventory/Deployed/pkg2/ID",
          "MOData": {
               "ID": "pkg2_id"
          }
     }
}
--simple_boundary--
```

The DM Server requests to send the current state of the Camera in the device to the Data Repository, and at the same time, the DM Server terminates the DM session. The Package#2 for this is as follows:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.oma.dm.request+json

{
     "CMD": [
          ["HPOST", "http://www.dr.com/camera_state?DevID=IMEI:493005100592800"
               "urn:oma:mo:oma-dcmo:1.0/(x)/Enabled?nv=(x)/Property:Camera"],
          ["END"]
     ]
}
```

Package#3 is not sent due to the END command. Instead, HTTP messages, that are not part of the DM session, are exchanged between the DM Client and the Data Repository. The HTTP POST request from the DM Client to the Data Repository is as follows:

```
POST /camera_state?DevID=IMEI:493005100592800 HTTP/1.1
Content-Type: text/plain
Host: www.dr.com

true
```

The HTTP response from the Data Repository to the DM Client is as follows:

```
HTTP/1.1 200 OK
```

I.2 Examples for Modifying MO data

In this section an example is presented in which the DM Server modifies several MO data in the device using the HGET command.

Package#1 that initializes the DM session is as follows (the DM Notification includes the REQ-MOS Option):

```
POST /dmclient/dm20 HTTP/1.1
Content-Type: application/vnd.oma.dm.initiation+json
Accept: application/vnd.oma.dm.request+json
OMADM-DevID: IMEI:493005100592800
Host: www.dms.com

{
     "MOS": [
          {
               "DDF": "http://www.vendor.com/DDF/devinfo1.0.ddf",
               "MOID": "urn:oma:mo:oma-dm-devinfo:1.0",
               "MIID": ["miid1"]
          },
          {
               "DDF": "http://www.vendor.com/DDF/oma-sessioninfomo1.0.ddf",
               "MOID": "urn:oma:mo:oma-sessioninfomo:1.0",
               "MIID": ["miid1"]
          },
          {
               "DDF": "http://www.vendor.com/DDF/oma-dm-dmacc2.0.ddf",
               "MOID": "urn:oma:mo:oma-dm-dmacc:2.0",
               "MIID": ["miid_dms1"]
          },
          {
               "DDF": "http://www.vendor.com/DDF/oma-dcmo1.0.ddf",
               "MOID": "urn:oma:mo:oma-dcmo:1.0",
               "MIID": []
          }
     ]
}
```

The DM Server updates the authentication secret for the DM Notification with the following Package#2:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.oma.dm.request+json

{
     "CMD": [
          ["HGET", "http://www.dms.com/new_secret?DevID=IMEI:493005100592800",
"urn:oma:mo:oma-dm-dmacc:2.0/(x)/AuthNoti/AuthSecret?nv=(x)/ServerID:DMS_ID1"]
     ]
}
```

After receiving the Package#2, the DM Client sends the HTTP GET request to the received ServerURI "http://www.dms.com/new_secret?DevID=IMEI:493005100592800" as follows:

```
GET /new_secret?DevID=IMEI:493005100592800 HTTP/1.1
Host: www.dms.com
```

The DM Client receives the following HTTP message as the response:

```
HTTP/1.1 200 OK
Content-Type: text/plain

AB123CES121XX90TPOWQESEFSE33
```

Note that above two HTTP messages are not part of the DM session. After the DM Client updates the *AuthNoti/AuthSecret* node in the DM Account MO with the received value "AB123CES121XX90TPOWQESEFSE33", the Package#3 is sent to the DM Server as follows:

```
POST /dmserver/dm20 HTTP/1.1
Content-Type: application/vnd.oma.dm.response+json
Accept: application/vnd.oma.dm.request+json
OMADM-DevID: IMEI:493005100592800
Host: www.dms.com

{
     "Status": [
          200
     ]
}
```

Now the DM Server sends the new DCMO instance to manage "USB" with the following Package#2:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.oma.dm.request+json

{
     "CMD": [
          ["HGET", "http://www.dms.com/dcmo_usb"]
     ]
}
```

After receiving the Package#2, the DM Client sends the HTTP GET request to the received ServerURI as follows:

```
GET /dcmo_camera HTTP/1.1
Host: www.dms.com
```

The DM Client receives the following HTTP message as the response:

```
HTTP/1.1 200 OK
Content-Type: application/dmmo+json

[
     {
          "DDF": "http://www.vendor.com/DDF/dcmo1.0.ddf",
          "ClientURI": "urn:oma:mo:oma-dcmo:1.0//",
          "MOData": {
               "usb": {
                    "Property": "USB",
                    "Group": "I/O",
                    "Description": "USB Control",
                    "Enabled": "true"
               }
          }
     }
]
```

After the DM Client creates the new DCMO instance, the Package#3 is sent to the DM Server as follows (the stored location for the new DCMO instance is delivered together with the status code):

```
POST /dmserver/dm20 HTTP/1.1
Content-Type: application/vnd.oma.dm.response+json
Accept: application/vnd.oma.dm.request+json
OMADM-DevID: IMEI:493005100592800
Host: www.dms.com

{
     "Status": [
          200, "urn:oma:mo:oma-dcmo:1.0/usb/"
     ]
}
```

The DM Server terminates the DM session with the following Package#2:

```
HTTP/1.1 200 OK
Content-Type: application/vnd.oma.dm.request+json

{
     "CMD": [
          ["END"]
     ]
}
```

I.3 Examples of DEFAULT Command

In this section two examples of Default Command usage are provided.

Example 1 (MMS):

DM Client receives the DEFAULT command with the following parameters:

- ServerURI: https://prov.dmserver.com/Settings/[MCC]/[MNC]/MMS
- MOID: urn:oma:mo:oma-mms:1.3

When the MMS Client starts up without configuration, it requests the DM Client to retrieve configuration related to the specific MOID. If no data are stored in the DM Tree for the requested MOID, the DM Client performs a HTTP GET to the provided ServerURI replacing "[MCC]" and "[MNC]" with real values. The retrieved data are stored at the location decided by the DM Client. The DM Server can retrieve the MO data sending the GET command against the ClientURI "urn:oma:mo:oma-mms:1.3/*/".

Example 2 (RCS):

DM Client receives the DEFAULT command with the following parameters:

- ServerURI: https://rcsprov.gsma.org/Settings/[MCC]/[MNC]/RCS
- MOID: urn:gsma:mo:rcs:rcse:1.0

When the RCS Client starts up, it triggers the DM Client to retrieve configuration. If no data are stored in the DM Tree, the DM Client performs a HTTP GET to the provided ServerURI replacing "[CMCC]" and "[CMNC]" with real values. The retrieved data are stored at the location decided by the DM Client. The DM Server can retrieve the MO data sending the GET command against the ClientURI "urn:gsma:mo:rcs:rcse:1.0/*/".