OMA Device Management Protocol

Candidate Version 1.3 – 25 May 2010

Open Mobile Alliance

OMA-TS-DM_Protocol-V1_3-20100525-C

Contents

Figures

Tables

1. Scope

This document describes a management protocol using the DM Representation Protocol [DMREPPRO]. This protocol is called the OMA Device Management Protocol, abbreviated as OMA DM Protocol, and it defines the protocol for various management procedures.

2. References

2.1 Normative References

[DMNOTI]	"OMA Device Management Notification Initiated Session, Version 1.3". Open Mobile Alliance™. OMA-TS-DM_Notification-V1_3. URL:http://www.openmobilealliance.org
[DMREPPRO]	"OMA Device Management Representation Protocol, Version 1.3". Open Mobile Alliance™. OMA-TS-DM_RepPro-V1_3. URL:http://www.openmobilealliance.org
[DMSEC]	"OMA Device Management Security, Version 1.3". Open Mobile Alliance™. OMA-TS-DM_Security-V1_3. URL:http://www.openmobilealliance.org
[DMSTDOBJ]	"OMA Device Management Standardized Objects, Version 1.3". Open Mobile Alliance™. OMA-TS-DM_StdObj-V1_3. URL:http://www.openmobilealliance.org
[DMTND]	"OMA Device Management Tree and Description, Version 1.3". Open Mobile Alliance™. OMA-TS-DM_TND-V1_3. URL:http://www.openmobilealliance.org
[IOPPROC]	"OMA Interoperability Policy and Process", Version 1.1, Open Mobile Alliance™, OMA-IOP-Process-V1_1, URL:http://www.openmobilealliance.org
[META]	"SyncML Meta Information, version 1.2". Open Mobile Alliance™. OMA-TS-SyncML_MetaInfo-V1_2. URL:http://www.openmobilealliance.org
[RFC2119]	"Key words for use in RFCs to Indicate Requirement Levels". S. Bradner. March 1997. URL:http://www.ietf.org/rfc/rfc2119.txt
[RFC2396]	"Uniform Resource Identifiers (URI): Generic Syntax". T. Berners-Lee, et al. August 1998. URL:http://www.ietf.org/rfc/rfc2396.txt
[RFC2616]	"Hypertext Transfer Protocol -- HTTP/1.1". R. Fielding, et al. June 1999. URL:http://www.ietf.org/rfc/rfc2616.txt

2.2 Informative References

None.

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.

Any reference to components of the DTD's or XML snippets is specified in this “`typeface`”.

3.2 Definitions

See the DM Tree and Description document [DMTND] for definitions of terms related to the management tree.

Full device URI	Full path to a device resource specified in the device's context. It is always a URI relative to the devices’ root node. Full device URI must always be used in the present specification.
Message	Atomic unit in OMA DM Protocol, one packet that the bearer network is able to accept. One OMA DM Protocol package could be divided into many messages.
Package	Package is a conceptual set of commands that could be spread over multiple messages.
Resource	A network data object or service that can be identified by a URI, as defined in Hypertext Transfer Protocol [RFC2616]. Resources may be available in multiple representations (e.g. multiple languages, data formats, size, and resolutions) or vary in other ways.

4. Introduction

The OMA Device Management Protocol allows management commands to be executed on nodes. It uses a package format defined in the DM Representation Protocol [DMREPPRO]. A node might reflect a set of configuration parameters for a device. Actions that can be taken against this node might include reading and setting parameter keys and values. Another node might be the run-time environment for software applications on a device. Actions that can be taken against this type of node might include installing, upgrading, or uninstalling software elements.

Actions are represented by OMA Device Management Protocol Commands, which are described in Device Management Representation Protocol [DMREPPRO].

5. Node addressing

DM supports two types of node addressing – absolute URI and relative URI. DM Clients and DM Servers MUST support absolute URI addressing and SHOULD support relative URI addressing.

5.1 Absolute URI Addressing

Each node MUST be addressed by a unique full device URI. URIs MUST follow requirements specified in Uniform Resource Identifiers (URI) [RFC2396] with the restrictions as specified in OMA Device Management Tree and Descriptions [DMTND]. Node addressing is defined in [DMTND].

Each node has a type that determines what kind of management content can be set/read on that object. Operations on a certain node require a predefined type of value to be sent and when the object is read, a value of that type is returned. For example a certain node can have a simple text type (text/plain) so simple text values can be set while another node stores complex types like the WAP Provisioning document type and require that value set in that node come with the WAP Provisioning document MIME type. Examples for other objects with complex values can be WAP settings or installed software.

In OMA DM Protocol, the target and source of a command are identified by the Target and Source elements respectively. Target refers to the recipient, and Source refers to the originator. Exceptions to this approach are mentioned in management commands requiring exception.

5.2 Relative URI Addressing

In case when the DM Server doesn't specify the absolute URI, the possibility is to address the node using relative URI as defined below. The relative URI contains two parts:

- Part A: This is used to identify the actual root URI of the Management Object that is to be managed. DM Client MUST resolve the actual path which begins from the root of Management Tree to the <x> node of the Management Object.
- Part B: This is used to identify the address of the node to be manipulated inside the Management Object, which starts after the root of the Management Object.

The syntax definition for Part A is ***URI*****?MOID=value&attribute=value** as described below:

- URI: The start point for the DM Client to find the MO occurrences in the whole sub-tree begins from 'URI'. This element SHOULD be included in the relative URI addressing.
- ?: This is the separator between the 'URI' and MOID. This element MUST be included in the relative URI addressing only when URI is present.
- MOID=value: This element is used to specify the MO Identifier which identifies the Management Object that is to be managed. This element MUST be included in the relative URI addressing. The character ":" MUST be percent encoded as "%3A" within MOID.
- &: This is the separator between the MOID and the attribute condition or between attribute conditions. This element MAY be included in the relative URI addressing.
- attribute=value: This is only used when the DM Server anticipates that multiple MO occurrences will be found. The <attribute> identifies the specific leaf node's URI relative to the root of the Management Object. The 'value' identifies the value of this leaf node. This part is used by the DM Client to find the unique MO occurrence that is to be managed. This element MAY be included in the relative URI addressing. If this element is specified, the preceding '&' MUST be specified as well. For reserved characters defined in section 3.4 "Query Component" of [RFC2396] MUST be percent encoded appropriately. If the DM Server need more attribute conditions to specify a unique MO, "&attribute=value" can occur from 0 time to N times with different attributes.

The Part A of relative URI MUST be included in the <TargetParent>/<LocURI> element. The Part B of relative URI MUST be included in the <Target>/<LocURI> element if it presents. For Add, Replace and Exec commands, the <TargetParent>

and <Target> elements MUST be included in the DM Message. For Delete command, the <TargetParent> element MUST be included in the DM Message, the <Target> element MAY be included in the DM Message.

The DM Client MUST resolve the relative URI to actual absolute URI if it supports relative URI addressing. The DM Client MUST find all the MO occurrences in the whole sub-tree according to the specified MO identifier. In case there are multiple MO occurrences found and “attribute=value” is provided by the DM Server, the DM Client MUST use the “attribute=value” to resolve the root URI of the unique MO occurrence that is to be managed.

In case the DM Server wishes to delete the whole MO occurrence(s), the Part B of relative URI is not needed. Then the Part A of relative URI will be enough to resolve the root URI(s) of the MO occurrence(s) to be removed from the Device. In this case <Target> element will not be included in <Delete> command of DM Message.

The actual URI MUST be constructed as the concatenation of resolved URI from Part A, followed by the “/” separator character, followed by the Part B of the relative URI.

There are two scenarios about how to address failure:

- Within Normal DM Session:
  - o If the DM Client doesn’t support this feature, the DM Client MUST return status code 406 (Optional Feature Not Supported).
  - o If the DM client failed to resolve the relative URI to actual absolute URI, the DM Client MUST return status code 400 (Bad Request).
  - o If the DM Client failed to find the unique MO occurrence if attribute=value is provided, the DM Client MUST return Status Code 404 (Not Found).
- Within Sessionless or Bootstrap Session:
  - o Since the response is not expected by the DM server, no status code will be returned. If the DM Client doesn’t support, or failed to resolve the URI to absolute URI, or failed to find the unique MO occurrence, the DM Client MUST NOT perform this command.

The relative URI concept is illustrated in the following diagram:

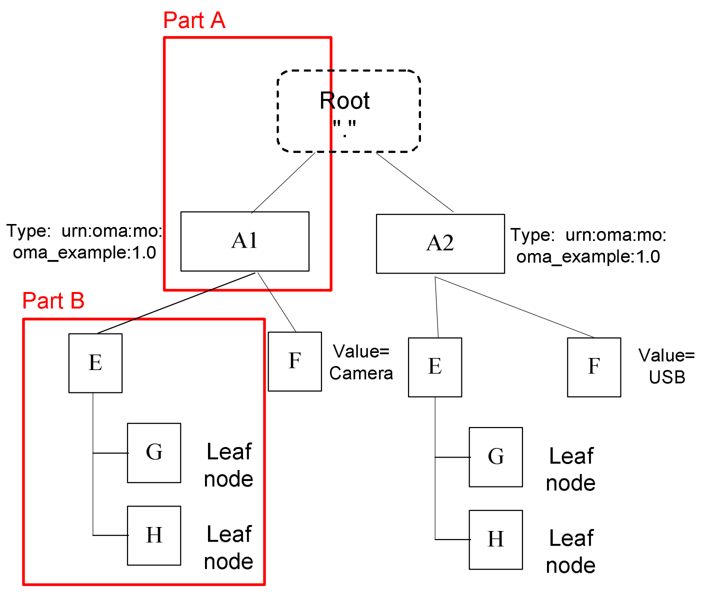


Figure 1: Relative URI Concept

In this example DM Server wants to manipulate node './A1/E/G'. The Part A of the relative URI sent by the DM Server is "`.?MOID=urn:oma:mo:oma_example:1.0&F=Camera`". The Part B of the relative URI sent by the DM Server is "E/G". The example message sent by DM Server is shown below:

```
<Exec>
  <CmdID>4</CmdID>
  <Item>
    <TargetParent>
      <LocURI>.?MOID=urn%3Aoma%3Amo%3Aoma_example%3A1.0&F=Camera</LocURI>
    </TargetParent>
    <Target>
      <LocURI>E/G</LocURI>
    </Target>
  </Item>
</Exec>
```

The resolved URI from Part A is "./A1" and the Part B is "E/G". Therefore the actual URI of node G is "./A1/E/G". Then the DM Client is able to execute the node './A1/E/G as expected by the DM Server.

The more complete relative URI concept is illustrated in the following diagram:

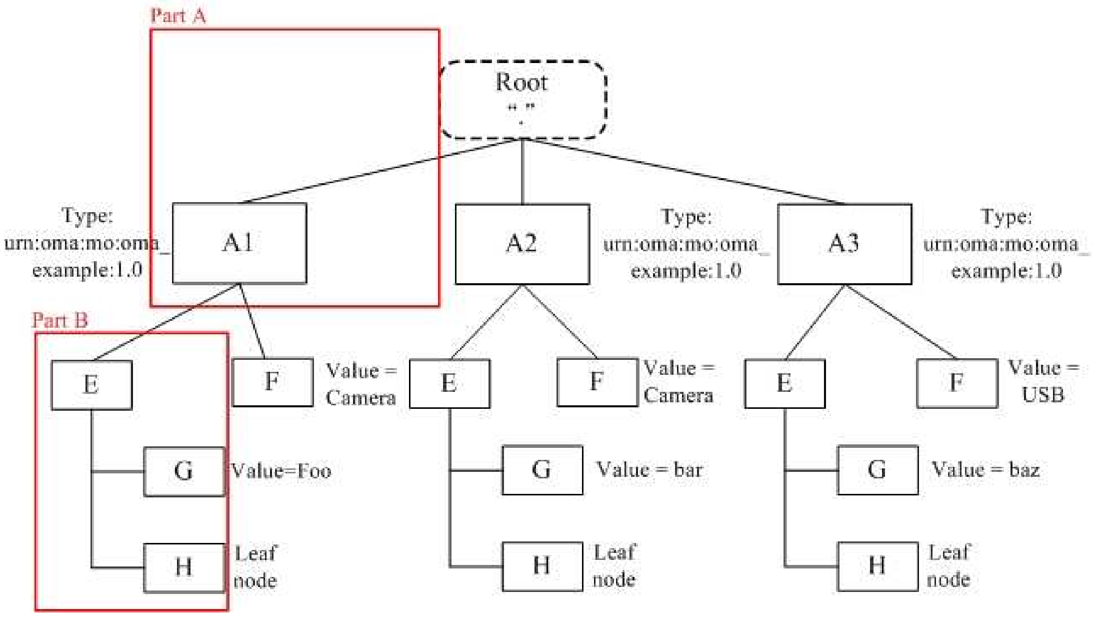


Figure 2: Relative URI Concept

In this example DM Server wants to manipulate node './A1/E/H'. The Part A of the relative URI sent by the DM Server is "`.?MOID=urn:oma:mo:oma_example:1.0&F=Camera&E/G=Foo`". The Part B of the relative URI sent by the DM Server is "E/H". The example message sent by DM Server is shown below:

```
<Exec>
  <CmdID>4</CmdID>
  <Item>
    <TargetParent>
      <LocURI>.?MOID=urn%3Aoma%3Amo%3Aoma_example%3A1.0&F=Camera&E/G=Foo</LocURI>
    </TargetParent>
    <Target>
      <LocURI>E/H</LocURI>
    </Target>
  </Item>
</Exec>
```

The resolved URI from Part A is "./A1" and the Part B is "E/H". Therefore the actual URI of node G is "./A1/E/H". Then the DM Client is able to execute the node './A1/E/H as expected by the DM Server.

6. Multiple Messages In Package

6.1 Description

The OMA Device Management protocol provides the functionality to transfer one SyncML package using multiple DM messages. This is necessary when one SyncML package is too large to be transferred in one SyncML message. For example, this limitation may be caused by the transport protocol or by the limitations of a small footprint device.

In OMA Device Management, the role of the package as a logical grouping of items is very limited. Most restrictions occur on messages, not on packages. For example, a command must fit entirely into one message. This includes the `Sequence` and `Atomic` commands, each of which must fit entirely into one message.

In order to avoid overwhelming a client with limited resources, a server is not permitted to send new commands to a client that has not yet returned a status to previous commands. In other words, most messages sent by the server to the client will correspond to a (one message) package, except in the case where a server is sending a large object or asking for more messages (using `Alert` 1222). A package containing a large object datum will span as many messages as necessary to transmit the large object, as specified in Section 7.

Note that the server is always in one of the following states with respect to package boundaries:

1. The server has sent a complete package. In this state, the server is awaiting status from the client on the commands sent in the package. Because the status and results may be large, such as the result of `Get` commands, the client may send multiple messages back to the server before completing its response.

2. The server has received a complete package (of responses) from the client. In this state, the server may send new commands to the client.

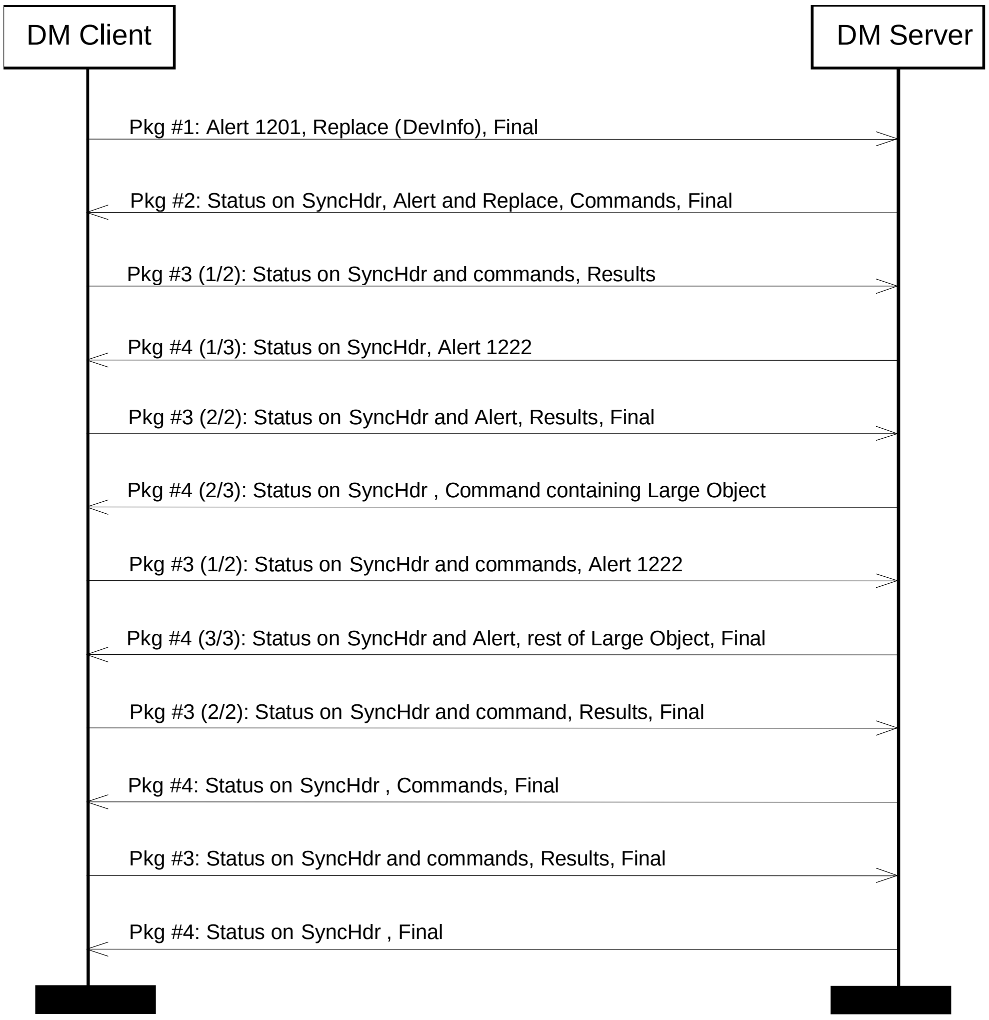
3. The server has sent one or more messages that are part of the same package, but has not yet sent the final message of the current package. This state is only valid when the server is sending a large object, and the package will end when the last chunk of the large object is sent.

Because the underlying transports for DM messages have a request/response form, either the client or the server may be required to send a message that contains neither new commands nor a `Final` flag, in order to keep the request/response cycle going.

For example, when the server is in State 1 (above), it may receive many messages from the client containing `Status` and `Results`. The server will respond to each such message sent by the client, but may not include new commands in those responses. Messages sent by the server in this state will contain a `Status` to the `SyncHdr` sent by the client and also the `Alert` 1222 (More Messages). `Status` MUST BE sent in response to `Alert` but MUST NOT be sent in response to `Results`.

It is also possible for `Alert` 1222 to be replaced by `Alert` 1223 (Session Abort) if the server wishes to abort the session.

The following chart shows an example of how multiple messages can be used.


DM Client
DM Server
Pkg #1: Alert 1201, Replace (DevInfo), Final
Pkg #2: Status on SyncHdr, Alert and Replace, Commands, Final
Pkg #3 (1/2): Status on SyncHdr and commands, Results
Pkg #4 (1/3): Status on SyncHdr, Alert 1222
Pkg #3 (2/2): Status on SyncHdr and Alert, Results, Final
Pkg #4 (2/3): Status on SyncHdr , Command containing Large Object
Pkg #3 (1/2): Status on SyncHdr and commands, Alert 1222
Pkg #4 (3/3): Status on SyncHdr and Alert, rest of Large Object, Final
Pkg #3 (2/2): Status on SyncHdr and command, Results, Final
Pkg #4: Status on SyncHdr , Commands, Final
Pkg #3: Status on SyncHdr and commands, Results, Final
Pkg #4: Status on SyncHdr , Final

6.2 Requirements

If a DM package is transferred in multiple DM messages, the last message in the package MUST include the `Final` element [DMREPPRO]. Other messages belonging to the package MUST NOT include the `Final` element.

The `Final` element MUST NOT be supplied by the client to close its package until the server has sent its `Final` element to close the previous package. For instance, the client MUST NOT supply the `Final` element to close package #2 or package #4 until the server has supplied the `Final` element which closes the previous package (#1 or #3, respectively). This is necessary because packages #2 and #4 constitute replies to the commands in packages #1 and #3.

The recipient of a DM package containing multiple messages MUST be able to ask for more messages. This is done by sending an `Alert` command, with the alert code 1222, back to the sender. If there are DM commands to be sent as a response to a preceding message, i.e. `Results`, the `Alert` command with the 1222 alert code MAY be omitted.

In the situation in which the server has sent the `Final` flag, and the client has not yet sent its `Final` flag, the server MUST respond to the client with the following "Next Message" response:

The "Next Message" response contains `Alert` code 1222 (or 1223 to abort), status to the `SyncHdr`, no other commands, and no `Final` flag.

A server MUST send the `Final` flag in every message, when possible. This is not possible during the sending of a Large Object (see Section 7), or when sending the "Next Message" response.

7. Large Object Handling

The protocol provides a means to synchronize an object whose size exceeds that which can be transmitted within one message. This is achieved by splitting the object into chunks that will fit within the message and using the <MoreData/> element to signal to the recipient that the data item is incomplete and has further chunks to come.

Clients SHOULD support Large Objects and servers MUST support Large Objects.

On receipt of a data object with the <MoreData/> element, the recipient MUST respond with a status response “213 – Chunked item accepted and buffered” and, if there are no other commands to be sent, ask for the next message using the Alert 1222 mechanism defined in section 6.

On receipt of the last chunk of the data object the recipient reconstructs the data object from its constituent chunks and applies the requested command. The appropriate status MUST then be sent to the originator. A command on a chunked object MUST implicitly be treated as atomic; i.e. the recipient can only commit the object once all chunks have been successfully received and reassembled.

Data objects that fit within a single message MUST NOT be followed by the <MoreData/> element. Data objects that span multiple messages MUST have the <MoreData/> element after all chunks except the last chunk.

A new data object MUST NOT be added by a sender to any message until the previous data object has been completed. If a data object is chunked across multiple messages the chunks MUST be sent in contiguous messages. New DM commands (i.e. Add, Replace, Delete, Copy, Atomic or Sequence) or new Items MUST NOT be placed between chunks of a data object.

Meta and Item information SHOULD be repeated on each subsequent message containing chunks of the same data object. Authentication details related to the data object MAY vary between messages bearing chunks of the same data object as defined in the section 9.

Clients that support Large Object Handling MUST indicate this by having the value of the DevDetail/LrgObj flag set to "true". The `MaxObjSize` accepted by the sender MAY be included in Meta information for the message header (`SyncHdr`) sent to the other party. `MaxObjSize` information sent in Meta information for the `SyncHdr` MUST be respected by the recipient, who MUST NOT send any single object larger than this size. If `MaxObjSize` is not sent, the recipient is free to send objects of any size back to the sender.

Note that the `MaxObjSize` remains in effect for the entire DM session, unless a new value is supplied in a subsequent message. A possible reason to send a new `MaxObjSize` in a later message in the same session might be that the `MaxObjSize` of a client might depend on free memory, which can decrease as objects are created and increase as objects are deleted. The `MaxObjSize` need not be a dynamic quantity, however.

If an item is chunked across multiple messages, the <Size> element of the Meta information MUST be used to signal to the recipient the overall size of the data object. The <Size> element MUST only be specified in the first chunk of the item.

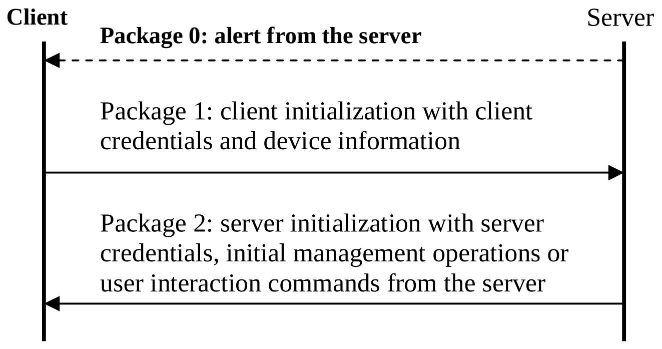
On receipt of the last chunk, the recipient MUST validate that the size of re-constituted chunks match the object <Size> supplied in the Meta information by the sender. If the size does not match then error status 424 “Size mismatch” MUST be returned. The recipient MUST NOT commit the command. The sender MAY attempt to retransmit the entire data object.

If the recipient detects a new data object or command before the previous item has been completed (by the chunk without the <MoreData/> Element), the recipient MUST respond with an Alert 1225 “End of Data for chunked object not received”. The Alert SHOULD contain the source and/or target information from the original command to enable the sender to identify the failed command. Note: a Status would not suffice here because there would not necessarily be a command ID to refer to. The recipient MUST NOT commit the command. The sender MAY attempt to retransmit the entire data object.

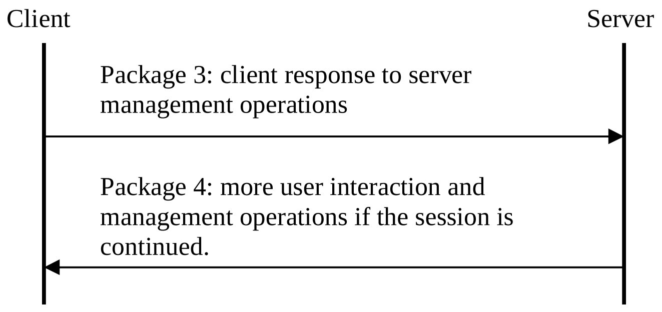
8. OMA DM Protocol packages

OMA DM Protocol consists of two parts: setup phase (authentication and device information exchange) and management phase. Management phase can be repeated as many times as the server wishes. Management sessions may start with Package 0 (the trigger). Trigger may be out-of-band depending on the environment and it is specified in DM Notification Initiated Session [DMNOTI].

The following chart depicts the two phases.


Client
Server
Package 0: alert from the server
Package 1: client initialization with client credentials and device information
Package 2: server initialization with server credentials, initial management operations or user interaction commands from the server

Setup phase


Client
Server
Package 3: client response to server management operations
Package 4: more user interaction and management operations if the session is continued.

Management phase

The Management Phase consists of a number of protocol iterations. The content of the package sent from the server to the client determines whether the session must be continued or not. If the server sends management operations in a package that need responses (`Status` or `Results`) from the client, the management phase of the protocol continues with a new package from client to server containing the client's responses to those management operations. The response package from client starts a new protocol iteration. The server can send a new management operation package and therefore initiate a new protocol iteration as many times as it wishes.

During the management phase when a package from server to client does not contain management operations or a challenge, the client will create a package containing only `Status` for `SyncHdr` as a response to the package received from server. In this case the entire response package MUST NOT be sent and the protocol ends. A server MUST send response packages to all client packages.

Processing of packages can consume unpredictable amount of time. Therefore the OMA DM Protocol does not specify any timeouts between packages.

If not enclosed by a Sequence or Atomic command, the client and server MAY freely choose the execution order of the management commands sent in the package. However, when execution order is required by the parent management command, commands MUST be executed in the order they were sent.

Client MUST NOT send any commands other than `Replace` command containing DevInfo or DevInfo, `Results` and `Alert` to the server.

8.1 Session Abort

8.1.1 Description

Either the client or the server may decide to abort the session at any time. Reasons for session abort may be server shutdown, client power-down, user interaction on the client, etc. In this case it is best if the aborting party sends a SESSION ABORT `Alert`. It is RECOMMENDED that the message also includes `Status` and `Results` of all the management commands that the aborting party executed before the abort operation.

If a recipient of a Session Abort sends a response to this message, the response is ignored.

Some cases of session aborts are not controllable, for example if the client goes out of coverage or its battery runs down. Servers and clients must be prepared for non-signalled session aborts as well. The requirements stated above are intended to reduce situations in which one party times out on a response from the other.

Implementations are possible (e.g. OBEX) in which the request/response roles of the transport binding may be reversed, i.e. the DM Client is a transport-level server, and the DM Server is a transport-level client. In this case, the recommendation in Section 8.1.1 above may not apply.

8.1.2 Requirement

`Alert` 1223 is used to signal an unexpected end to the device management session. The sender of the Session Abort alert MAY also include `Status` and `Results` of all the management commands that the aborting party executed before the abort operation. The sender MUST include a `Final` flag. A server receiving this alert SHOULD respond with a message that MUST contain status for the `Alert` and the `SyncHdr` and no new commands.

A client receiving `Alert` 1223 SHOULD NOT respond.

8.2 Package 0: Management Initiation Alert from server to client

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”.

A management server can use this notification capability to cause the client to initiate a connection back to the management server. OMA DM Protocol specifies several Management Initiation notification bearers. Definition of bearers and notification content can be found from [DMNOTI] specification.

Note that an identical effect to receiving a Management Initiation notification can be caused in other ways. For example, the user interface (UI) of the device may allow the user to tell the client to initiate a management session. Alternatively, the management client might initiate a session as the result of a timer expiring. A fault of some type in the device could also cause the management client to initiate a session.

8.3 Package 1: Initialization from client to server

The setup phase is virtually identical to that described in the [SYNCPRO]. The purpose of the initialization package sent by the client is:

- To send the DevInfo information (like manufacturer, model etc) to a Device Management Server as specified [DMSTDOBJ]. Client MUST send DevInfo information in the first message of management session.
- To send the DevDetail information (that is specified in [DMSTDOBJ]) to a Device Management Server, if it is requested in Package 0 message (as specified in [DMNOTI]).

- To identify the client to the server according to the rules specified in Section 9.
- To inform the server whether the management session was initiated by the server (by sending a trigger in Package 0) or by the client (like end user selecting a menu item).
- To inform the server of any optional Client generated alert, for example Generic Alert or Client Event [DMREPPRO].

The detailed requirements for the initialization package from the client to server (Package 1) are:

1. The requirements for the elements within the `SyncHdr` element.
    - The value of the `VerDTD` element MUST be '1.2'.
    - The value of the `VerProto` element MUST be 'DM/1.3'.
    - `SessionID` MUST be included to indicate the ID of the management session. If the client is responding to notification, with alert code SERVER-INITIATED MGMT (1200), then `SessionID` MUST be same as in notification. Otherwise, the client generates a `SessionID` which should be unique for that client. The same `SessionID` MUST be used throughout the whole session.
    - `MsgID` MUST be used to unambiguously identify the message belonging to the management session from server to client.
    - The `Target` element MUST be used to identify the target server.
    - The `Source` element MUST be used to identify the source device.
    - The `Cred` element MAY be included in the authentication message from the Device Management client to Device Management server as specified in Section 9.
2. `Alert` MUST be sent whether the client or the server initiated the management session in the `SyncBody`. The requirement for the `Alert` command follows:
    - `CmdID` is REQUIRED.
    - The `Data` element is used to carry the management session type which can be either SERVER-INITIATED MGMT (1200) or CLIENT-INITIATED MGMT (1201).
3. The DevInfo information MUST be sent using the `Replace` command in the `SyncBody`. The requirement for the `Replace` command follows:
    - `CmdID` is REQUIRED.
    - An `Item` element per node found from DevInfo tree. Possible nodes in DevInfo tree are specified in [DMSTDOBJ].
    - The `Source` element in the `Item` element MUST have a value indicating URI of node.
    - The `Data` element is used to carry the DevInfo data.
4. The DevDetail information MUST only be sent using the Replace command in the SyncBody if requested by the DM Server. The requirement for the Replace command follows:
    - `CmdID` is REQUIRED.
    - An `Item` element per node found from DevDetail tree. Possible nodes in DevDetail tree are specified in [DMSTDOBJ].

    - The `Source` element in the `Item` element MUST have a value indicating URI of node.
    - The `Data` element is used to carry theDevDetail data.

5. Client MAY include client-generated alerts such as Client Event [DMREPPRO] or Generic Alert.

The `Final` element MUST be used in the `SyncBody` for the message, which is the last in this package.

8.4 Package 2: Initialization from server to client

The purpose of the initialization package sent by the server is to:

- Identify the server to the client according to the rules specified in Section 9.
- Optionally, the server can send user interaction commands.
- Optionally to send management data and commands.
- Send status of Client Initiated Alerts if any of these was received from the client

Package 2 MAY close the management session by containing only the <Final> element (any management command, user interaction command or client authentication challenge will continue the session). Alternately, the server may send the Session Abort Alert (1223) to force the close of the session in extreme situations.

The detailed requirements for package 2 are:

- o The requirements for the elements within the `SyncHdr` element.
    - The value of the `VerDTD` element MUST be '1.2'.
    - The value of the `VerProto` element MUST be ‘DM/1.3’ when complying with this specification.
    - `SessionID` MUST be included to indicate the ID of the management session.
    - `MsgID` MUST be used to unambiguously identify the message belonging to the management session from server to client.
    - The `Target` element MUST be used to identify the target device.
    - The `Source` element MUST be used to identify the source device.
    - `Cred` element MAY be included in the authentication message according to the rules described in Section 9. Server is always authenticated to the device but this authentication MAY be accomplished at the transport level.
- o The `Status` MUST be returned in the `SyncBody` for the `SyncHdr` and `Alerts` sent by the client.
- o Any management operation including user interaction in the DM document (e.g. `Alert`, `Sequence`, `Replace`) are placed into the `SyncBody`.
    - `CmdID` is REQUIRED.
    - `Source` MUST be used if URI is needed to further address the source dataset.
    - `Target` MUST be used if URI is needed to further address the target dataset.
    - The `Data` element inside `Item` is used to include the data itself unless the command does not require a `Data` element.

    - The `Meta` element inside an operation or inside an `Item` MUST be used when the `Type` or `Format` are not the default values [META].

  - The `Final` element MUST be used in the `SyncBody` for the message, which is the last in this package.

8.5 Package 3: Client response sent to server

The content of package 3 is:

- Results of management actions sent from server to client.
- Results of user interaction commands.
- New optional Client generated alert, for example Generic Alert or Client Event [DMREPPRO] that was raised during the session.

This package is sent by the client if Package 2 contained management commands that required a response from the client.

The detailed requirements for package 3 are:

1. The requirements for the elements within the `SyncHdr` element.
    - The value of the `VerDTD` element MUST be '1.2'.
    - The value of the `VerProto` element MUST be 'DM/1.3'.
    - `SessionID` MUST be included to indicate the ID of the management session.
    - `MsgID` MUST be used to unambiguously identify the message belonging to the management session from server to client.
    - The `Target` element MUST be used to identify the target device.
    - The `Source` element MUST be used to identify the source device.
2. `Status` MUST be returned for the `SyncHdr` and `Alert` command sent by the device management server in the `SyncBody`.
3. `Status` MUST be returned in the `SyncBody` for management operations sent by the server in Package 2.
4. `Results` MUST be returned in the `SyncBody` for successful `Get` operations sent by the server in the previous package and the following requirements apply:
    - `Results` MUST contain `Meta` element with `Type` and `Format` elements describing content of `Data` element, unless the `Type` and `Format` have the default values [META].
    - Items in Results MUST contain the `Source` element that specifies the source URI.
5. Client MAY send client generated alerts, for example Client Event [DMREPPRO] or Generic Alert.

The `Final` element MUST be used in the `SyncBody` for the message, which is the last in this package.

8.6 Package 4: Further server management operations

Package 4 is used to close the management session. If the server sends any operation in Package 4 that needs response from the client, the protocol restarts from Package 3 with a new protocol iteration. Server sends results of Client Initiated Alerts if any of these was received from the client in previous package. The detailed requirements for package 4 are:

1. The requirements for the elements within the `SyncHdr` element.

- The value of the `VerDTD` element MUST be '1.2'.
- The value of the `VerProto` element MUST be ‘DM/1.3’.
- `SessionID` MUST be included to indicate the ID of the management session.
- `MsgID` MUST be used to unambiguously identify the message belonging to the management session from server to client.
- The `Target` element MUST be used to identify the target device.
- The `Source` element MUST be used to identify the source device.

2. `Status` MUST be returned for the `SyncHdr` sent by the device management server in the `SyncBody` and if any Alerts were sent by the client then the server MUST send Status for these Alerts.
3. Any management operation including user interaction in the DM document (e.g. `Alert`, `Sequence`, `Replace`) placed into the `SyncBody`.
   - `CmdID` is REQUIRED.
   - `Source` MUST be used if URI is needed to further address the source dataset.
   - `Target` MUST be used if URI is needed to further address the target dataset.
   - The `Data` element inside `Item` is used to include the data itself unless the command does not require a `Data` element.
   - The `Meta` element inside an operation or inside an `Item` MUST be used when the `Type` or `Format` are not the default values [META].

The `Final` element MUST be used in the `SyncBody` for the message, which is the last in this package. Package 4 MAY close the management session by containing only the <Final> element (any management command or user interaction command will continue the session). Alternately, the server may send the Session Abort Alert (1223) to force the close of the session in extreme situations.

8.7 Generic Alert

The protocol defines a Generic Alert message for Alerts generated by the client that MAY have a relation to a Management Object. In the case of a relation to a Management Object then the Source and LocURI MUST identify the address to that Management Object.

Anytime after the Client or Server Initiated Management Alert, the client MAY send a Generic Alert message to the server. The Generic Alert message SHALL only be sent from the client to the server. After the server has received the Generic Alert the server MUST respond with the status for how the server handles all Items.

The client MAY send multiple Alert messages of code “Generic Alert” or combine them together with multiple Items inside one or multiple Alert message of code “Generic Alert”. The Data in the Generic Alert message is not specified in the protocol, the protocol will specify how the client can inform the server what Type and Format it is. The server MUST support the Generic Alert Format but not all Types of the alert data. The Server MUST respond with status 415 “Unsupported media Type or Format” if the Type and Format are unsupported by the server. If the device does not support Large Object then the Alert message MUST NOT exceeds the message size.

This specification only specifies what is required from the protocol perspective, some registered Generic Alerts MAY have additional requirements for different Types of Generic Alert Data. For example, one registered Type of Alert may define that a reference to a Management Object is mandatory and the Format must be of Type Integer and the Alert Data must be included inside the Data.

The optional parameter Mark MUST contain the importance level. If the parameter is omitted then the default importance level is assumed.

The server MUST respond with status 200 “OK” or 202 “Accepted for processing” if the server has received the Alert without any errors and is capable of processing the Data in the Alert. In other cases the server MUST use one of the following error status codes: 401, 406, 407, 412, 415 or 500.

8.7.1 Generic Alert Message

This is the basic design of a Generic Alert message:

```
<Alert>
   <CmdID>2</CmdID>
   <Data>1226</Data>      <!-- Generic Alert -->
   <Correlator>abc123</Correlator>
   <Item>
      <Source><LocURI>./SyncML/Sample</LocURI></Source>
      <Meta>
         <Type xmlns='syncml:metinf'>
            Reversed-Domain-Name: org.domain.samplealert
         </Type>
         <Format xmlns='syncml:metinf'>xml</Format>
         <Mark xmlns='syncml:metinf'>critical</Mark>   <!-- Optional -->
      </Meta>
      <Data>
         <!-- Client Alert Data Goes Here -->
      </Data>
   </Item>
</Alert>
```

8.7.1.1 CmdID

This MUST be specified in the same way as all commands.

8.7.1.2 Data

This MUST be specified with the value 1226 [DMREPPRO] for Generic Alert.

8.7.1.3 Item

This is a REQUIRED parameter. Item MUST be repeated for each alert of type Generic Alert if the device will send them together inside the same Alert message.

8.7.1.4 LocURI within Source

This is an optional parameter. If the Alert is generated from a Management Object and the definition of that Management Object mandates this parameter, then it MUST be included.

8.7.1.5 Meta

The Meta element MUST be specified for the Type and Format of the Alert Data.

8.7.1.6 Type

The Type element MUST be specified and specifies the media type of the content information in the Data element. The content information for this element type MUST be a URN. If it is a MIME-type, then it MUST use “Content-Type” as namespace identifier and the content SHOULD be a registered MIME content-type. If it is a reverse domain name then the namespace identifier “Reversed-Domain-Name” MUST be specified. No other namespace identifiers than these two are allowed. The ABNF syntax for the content of reverse domain name MUST be:

<URN> ::= <domain> [/<name>]*

<domain> ::= “reversed domain name”

<name> ::= alphanum

NOTE: For simplicity and portability, each name element is currently restricted to alphanumeric characters. (Alphanum is defined in [RFC2396].)

Example of two valid alert types:

- “Content-Type: application/samplealert”
- “Reversed-Domain-Name: org.openmobilealliance.dm.samplealert”

8.7.1.7 Format

The Format element MUST be specified. Format MUST contain a DM identifier of the Format of the following Data element.

8.7.1.8 Mark

The Mark element MAY be specified. Mark will define the importance level of the alert message. The following levels are allowed in Generic Alert: fatal, critical, minor, warning, informational, harmless and indeterminate. There the order indicates the importance level with fatal as most important and indeterminate as least important. If the Mark element is omitted then the default importance level “informational” is assumed.

8.7.1.9 Data (inside <Item>)

The Data element MUST be specified. Data MUST use the Format and Type specified in the Meta tag.

8.7.1.10 Correlator

The Correlator is an optional field and is used when the alert is an asynchronous response to an Exec command. Typically, the Correlator field in the alert echoes the Correlator value from an Exec command and is omitted in all other instances. Registered Generic Alerts SHOULD specify how the Correlator field is used.

9. Authentication

OMA DM Protocol uses the authentication framework specified in this chapter, with extensions defined in OMA Device Management Security [DMSEC]. This section specifies the rules for how the OMA-DM Protocol-level and the transport-level authentication are used.

Server and client can both challenge each other if no credentials were given in the original request or the credentials were considered too weak. If the server sent no credentials or invalid credentials in Package #2, no challenge and no commands (only Status to SyncHdr and DevInfo), the client MUST NOT challenge the server by sending back only a Status for the SyncHdr with a challenge. If the server challenged the client in Pkg 2, the client MUST revert to pkg 1 and MUST resend the Alert and DevInfo along with the credentials requested by the server.

The preferred authentication type of the server may be indicated to the client using the <X>/AAuthPref parameter in DM Account management object [DMSTDOBJ].

Generation and maintenance of client and server credentials are out of scope of the OMA DM Protocol specification.

In this chapter, the authentication procedures are defined for the basic and MD5 digest access authentication.

9.1 Authentication Challenge

If the response code to a request (message or command) is 401 (‘Unauthorized’) or 407 (‘Authentication required’), the request requires authentication. In this case, the Status command to the request MUST include a Chal element (See [DMREPPRO]). The Chal contains a challenge applicable to the requested resource. The originator MAY repeat the request with a suitable Cred element (See [DMREPPRO]). If the request already included the Cred element, then the 401 response indicates that authorization has been refused for those credentials.

Both the client and the server can challenge for authentication.

If the 401 response (i.e., Status) contains the same challenge as the prior response, and the user agent has already attempted authentication at least once, then the user SHOULD be presented the entity that was given in the response, since that entity might include relevant diagnostic information.

If the response code to a request is 212 (‘Authentication accepted’), no further authentication is needed for the remainder of the DM session. In the case of the MD5 digest access authentication, the Chal element can however be returned. Then, the next nonce in Chal MUST used for the digest when the next DM session is started.

If a request includes security credentials and the response code to the request is 200, the same credentials MUST be sent within the next request. If the Chal element is included and the MD5 digest access authentication is mandated, a new digest is created by using the next nonce. In the case of the MD5 digest access authentication, the Chal element can however be returned. The next nonce in Chal MUST be used when the next request is sent.

Once authentication has occurred, the authentication type for a security layer MUST be kept same for the whole session.

In case of authentication failure (either the userid and/or password was wrong or authentication was mandated) requirements are:

- The response message indicating the authentication failure on application layer (see chapter 9.3) contains only Status commands (i.e. Replace, Get etc. commands MUST NOT be specified in the response). A Status command MUST be provided for every command received in the request.
- In case the session is continued, the next message containing the proper credentials MUST contain a Status for the SyncHdr, MUST have the same SessionID as the previous messages and the message MUST be sent to the RespURI, if it was specified in the response indicating the authentication failure.

9.2 Authorization

The Cred element MUST be included in requests (message or command), which are sent after receiving the 401 or 407 responses if the request is repeated. In addition, it can be sent in the first request from a device if the authentication is mandated through pre-configuration. The content of the Cred element is specified in [DMREPPRO]. The authentication type is dependent on the challenge (See the previous chapter) or the pre-configuration.

9.3 Application Layer Authentication

The authentication on the application layer is accomplished by using the Cred element in SyncHdr and the Status command associated with SyncHdr. Within the Status command, the challenge for the authentication is carried as defined earlier. The authentication can happen both directions, i.e., the client can authenticate itself to the server and the server can authenticate itself to the client.

9.4 Authentication Examples

9.4.1 Basic authentication with a challenge

At this example, the client tries to initiate with the server without any credentials (Pkg #1). The server challenges the client (Pkg #2) for the application layer authentication. The client MUST send Pkg #1 again with the credentials. The server accepts the credentials and the session is authenticated (Pkg #2). In the example, commands in SyncBody are not shown although in practice, they would be there.

Pkg #1 from Client:

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.3</VerProto>
    <SessionID>1</SessionID>
    <MsgID>1</MsgID>
    <Target><LocURI>http://www.syncml.org/mgmt-server</LocURI></Target>
    <Source><LocURI>IMEI:493005100592800</LocURI></Source>
  </SyncHdr>
  <SyncBody>
    ...
  </SyncBody>
</SyncML>
```

Pkg #2 from Server:

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.3</VerProto>
    <SessionID>1</SessionID>
    <MsgID>1</MsgID>
    <Target><LocURI>IMEI:493005100592800</LocURI></Target>
    <Source><LocURI>http://www.syncml.org/mgmt-server</LocURI></Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <CmdID>1</CmdID>
      <MsgRef>1</MsgRef>
      <CmdRef>0</CmdRef>
      <Cmd>SyncHdr</Cmd>
      <TargetRef>http://www.syncml.org/mgmt-server</TargetRef>
      <SourceRef>IMEI:493005100592800</SourceRef>
      <Chal>
        <Meta>
          <Type xmlns='syncml:metinf'>syncml:auth-basic</Type>
          <Format xmlns='syncml:metinf'>b64</Format>
        </Meta>
      </Chal>
      <Data>407</Data> <!-- Credentials missing -->
    </Status>
    ...
  </SyncBody>
</SyncML>
```

Pkg #1 (with credentials) from Client:

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.3</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target><LocURI>http://www.syncml.org/mgmt-server</LocURI></Target>
    <Source><LocURI>IMEI:493005100592800</LocURI></Source>
    <Cred>
      <Meta>
      <Type xmlns='syncml:metinf'>syncml:auth-basic</Type>
      <Format xmlns='syncml:metinf'>b64</Format>
      </Meta>
      <Data>QnJ1Y2UyOk9oQmVoYXZl</Data>
      <!-- base64 formatting of 'userid:password' -->
    </Cred>
  </SyncHdr>
  <SyncBody>
    ...
  </SyncBody>
</SyncML>
```

Pkg #2 from Server:

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.3</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target><LocURI>IMEI:493005100592800</LocURI></Target>
    <Source><LocURI>http://www.syncml.org/mgmt-server</LocURI></Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <CmdID>1</CmdID>
      <MsgRef>2</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
      <TargetRef>http://www.syncml.org/mgmt-server</TargetRef>
      <SourceRef>IMEI:493005100592800</SourceRef>
```

```
        <Data>212</Data> <!-- Authenticated for session -->
      </Status>
      ...
    </SyncBody>
</SyncML>
```

9.4.2 MD5 digest access authentication with a challenge

At this example, assume (as in 9.4.1 above) the client tries to initiate with the server without any credentials (Pkg #1 is omitted here for brevity). The server challenges the client as above (Pkg #2 is also omitted from this example) for the application layer authentication. The authentication type is now syncml:auth-md5 (MD5 digest access authentication). The client MUST resend Pkg #1 this time with the MD5 credentials (as shown below in Pkg #1). The server accepts the credentials and the session is authenticated (Pkg#2 below). Also, the server sends the next nonce to the client, which the client MUST use when the next DM session is started. In the example, commands in SyncBody are not shown although in practice, they would be there.

Pkg #1 from Client:

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.3</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target><LocURI>http://www.syncml.org/mgmt-server</LocURI></Target>
    <Source>
      <LocURI>IMEI:493005100592800</LocURI>
    <LocName>Bruce2</LocName>    <!-- userid -->
    </Source>
    <Cred>
      <Meta>
       <Type xmlns='syncml:metinf'>syncml:auth-md5</Type>
       <Format xmlns='syncml:metinf'>b64</Format>
      </Meta>
    <Data>Zz6EivR3yeaaENcRN6lpAQ==</Data>
    <!-- Base64 coded MD5 for user 'Bruce2', password 'OhBehave', nonce
    'Nonce' -->
    </Cred>
  </SyncHdr>
  <SyncBody>
    ...
  </SyncBody>
</SyncML>
```

Pkg #2 from Server:

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.3</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target><LocURI>IMEI:493005100592800</LocURI></Target>
    <Source><LocURI>http://www.syncml.org/mgmt-server</LocURI></Source>
```

```
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>1</MsgRef>
         <CmdRef>0</CmdRef>
         <Cmd>SyncHdr</Cmd>
         <TargetRef>http://www.syncml.org/mgmt-server</TargetRef>
         <SourceRef>IMEI:493005100592800</SourceRef>
         <Chal>
            <Meta>
               <Type xmlns='syncml:metinf'>syncml:auth-md5</Type>
               <Format xmlns='syncml:metinf'>b64</Format>
               <NextNonce xmlns='syncml:metinf'>LG3iZQhhdmKNHg==</NextNonce>
               <!-- This nonce is used at the next session -->
            </Meta>
         </Chal>
         <Data>212</Data> <!-- Authenticated for session -->
      </Status>
      ...
   </SyncBody>
</SyncML>
```

10.User interaction commands

10.1 Introduction

The OMA Device Management Protocol specifies following user interaction types for example to notify and obtain confirmation from the user regarding the management operation. These interaction types are the following:

- User displayable notification associated with a certain action.
- Confirmation from the user to execute a certain management operation.
- Prompt user to provide input for upcoming management operation.
- Prompt user to select item or items among items.
- Display progress notification for a certain action.

10.2 User interaction alert codes

These `Alert`s can be sent only from the server to the client. Clients MUST report 406: “Optional Feature Not Supported”, if client does not support User Interaction Alerts. If sent by the client, they are ignored by the server. Multiple user interaction `Alert`'s can be present in Package 2, in this case the client executes them by arbitrary order (unless `Sequence` is used) and sends back the results in multiple `Status` packages in Package 3. If the protocol continues after Package 4, Package 4 can also contain user interaction `Alert`'s.

When a user interaction is executed, server is notified about the outcome of the interaction in a `Status` message. The user interaction-specific `Status` responses are described in [DMREPPRO].

All user interaction `Alert`s contain two or more `Item` elements. Client MUST preserve the order of these `Item` elements. Client MUST also process these `Item` elements in the same order as they are in the message.

User interactions, except display, SHOULD have user option to cancel operation. If the user decides to cancel the operation, then management message processing is stopped. Status codes for executed commands are reported normally and status code `(215) Not executed` is returned to all commands which are not processed. After processing the user response the server might decide to continue protocol with some other management operation.

If the UI allows the user to cancel (for any of the UI `Alerts`), then the status `(214) Operation cancelled` should be returned for the `Alert`.

10.2.1 Display

The DM DISPLAY `Alert` is slightly changed in OMA DM Protocol. The `Alert` has two Items.

- The first `Item` contains optional parameters as specified in Section 10.3.
- The second `Item` has exactly one `Data` element containing the text to be displayed to the user.

Example:

```
<Alert>
  <CmdID>2</CmdID>
  <Data>1100</Data>
  <Item><Data>MINDT=10</Data></Item>
  <Item>
    <Data>Management in progress</Data>
  </Item>
</Alert>
```

10.2.2 Confirmation

Confirmation is a binary decision: the user either approves or rejects the option. A new `Alert` code is introduced for this purpose, the CONFIRM_OR_REJECT. When the client receives this `Alert`, it displays the `Alert` text then enables the user to select "Yes" or "No". If the answer is "Yes", the status code 200: “Yes” is returned and the processing of the package continues without change in processing. If the answer is "No", the status code 304: “No” is returned and the processing of the package ceases. If the UI allows the user to cancel, status 214: “Operation cancelled” should be returned for the Alert.

If user answers "No", then package processing will change according to placement of confirmation `Alert` in package as follows.

- If confirmation `Alert` is inside `Atomic`, then `Atomic` fails and all executed commands have to be rolled back.
- If confirmation `Alert` is inside `Sequence`, then commands in `Sequence` after confirmation `Alert` are bypassed.
- If confirmation `Alert` is not inside `Atomic` or `Sequence` i.e. it is directly in `SyncBody`, then user response has no effect to package processing. In this way server can query user opinion before sending actual management commands to client device.

Status code `(215) Not Executed` will be sent back for the commands whose execution was bypassed as result of user interaction.

The `Alert` contains two `Item`s.

- The first `Item` contains the optional parameters as specified in Section 10.3.
- The second `Item` has exactly one `Data` element containing the text to be displayed to the user.

Example:

```
<Alert>
  <CmdID>2</CmdID>
  <Data>1101</Data>
  <Item></Item> <!-- no optional parameters -->
  <Item>
    <Data>Do you want to add the CNN access point?</Data>
  </Item>
</Alert>
```

Result if user responds "No":

```
<Status>
  <CmdID>2</CmdID>
  <MsgRef>1</MsgRef>
  <CmdRef>2</CmdRef>
  <Cmd>Alert</Cmd>
  <Data>304</Data> <!-- Not modified -->
</Status>
```

If the result in the above example had been that the user chose Yes, the status would have been `(200)`.

10.2.3 User input

When this `Alert` is sent, the client displays the text then allows the user to type in a text string. This text string is then sent back to the server in a `Status` message.

The server instructs the client to execute this user interaction by sending a TEXT INPUT `Alert`. The `Alert` contains at least two `Items`.

- The first `Item` contains optional parameters as specified in Section 10.3.
- The second `Item` has exactly one `Data` element containing the text to be displayed to the user.

Example:

```
<Alert>
   <CmdID>2</CmdID>
   <Data>1102</Data>
   <Item></Item>
   <Item>
      <Data>Type in the name of the service you would like to
configure</Data>
   </Item>
</Alert>
```

The user is presented with the text and an input box to type in the message. The following `Status` message is sent back in the next message from client to server:

```
<Status>
   <MsgRef>1</MsgRef>
   <CmdRef>2</CmdRef>
   <Cmd>Alert</Cmd>
   <Data>200</Data> <!-- Successful, user typed in a text -->
   <Item>
      <Data>CNN</Data> <!-- User input -->
   </Item>
</Status>
```

10.2.4 User choice

When this `Alert` is sent, the user is presented with a set of possible choices. The `Alert` body MUST contain the following `Items`.

- The first `Item` contains optional parameters as specified in Section 10.3.
- The second `Item` has exactly one `Data` element containing the title of the selection as plain text.
- From third `Item` onwards the Item contains exactly one `Data` element that describes one possible choice as plain text. These `Items` are referenced by a number starting from 1. `Items` MUST be numbered in the order they were sent. `Items` SHOULD be presented to the user in the order they were sent.

The user selection is returned in `Status` message. The selected item is returned in an `Item`. The `Data` element of this `Item` contains the reference number of item. A variation of this `Alert` allows the user to select multiple items. In this case selected items are sent back in multiple `Items` in the same way as one selected item.

One possible implementation could be a list and each `Data` member of the `Alert` could be displayed as a row in the list. The user could select a list item then he or she would push the "Ok" button and the ID of the selected list item is sent back in a `Status` message.

Example for a single-choice `Alert`:

```
<Alert>
```

```
   <CmdID>2</CmdID>
   <Data>1103</Data>
   <Item><Data>MINDT=10</Data></Item>
   <Item>
      <Data>Select service to configure</Data>
   </Item>
   <Item>
      <Data>CNN</Data>
   </Item>
   <Item>
      <Data>Mobilbank</Data>
   </Item>
   <Item>
      <Data>Game Channel</Data>
   </Item>
</Alert>
```

Response to this `Alert` returns the selected item.

```
<Status>
   <MsgRef>1</MsgRef>
   <CmdRef>2</CmdRef>
   <Cmd>Alert</Cmd>
   <Data>200</Data> <!-- Successful, user selected an item -->
   <Item>
      <Data>2</Data> <!-- User selected MobilBank -->
   </Item>
</Status>
```

Example to multiple-choice `Alert`

```
<Alert>
   <CmdID>2</CmdID>
   <Data>1104</Data>
   <Item></Item>
   <Item><Data>Select service to configure</Data></Item>
   <Item>
      <Data>CNN</Data>
   </Item>
   <Item>
      <Data>Mobilbank</Data>
   </Item>
   <Item>
      <Data>Game Channel</Data>
   </Item>
</Alert>
```

Response to this `Alert` returns the selected item. The number of the selected items can be returned in arbitrary order by the client.

```
<Status>
   <MsgRef>1</MsgRef>
   <CmdRef>2</CmdRef>
   <Cmd>Alert</Cmd>
   <Data>200</Data> <!-- Successful, user selected an item -->
   <Item>
      <Data>3</Data>
   </Item>
   <Item>
      <Data>2</Data> <!-- User selected Mobilbank and Game Channel -->
   </Item>
</Status>
```

10.2.5 Progress notification (object download)

Users SHOULD be able to track the progress of a longer management operation like a file or object download. OMA Device Management Protocol will not provide a separate mechanism for progress notification but it will entirely reuse the DM `Size` Meta-Information tag defined in DM Meta-Information DTD [META] and will make a recommendation for device manufacturers to use this tag for displaying progress notification.

According to DM Meta-Information DTD, any `Item` can be tagged by `Size` meta-information that indicates the size of the object. When the device encounters a `Size` meta-information tag in a received `Item`, it MAY display a progress notification on the user interface if the device decides that the item with the given size will take a longer time to download. The progress notification bar is scaled according to the length information conveyed in the `Size` element. If the size information is not sent by the server, the client is not able to display a scaled progress bar so it is recommended that servers send this information if the object to be downloaded by the client is reasonably large.

Example of an antivirus data file download with `Size` Meta Information.

```
<Add>
   <CmdID>2</CmdID>
   <Meta>
      <Format xmlns='syncml:metinf'>b64</Format>
      <Type xmlns='syncml:metinf'>
         application/antivirus-inc.virusdef
      </Type>
   </Meta>
   <Item>
      <Meta>
         <!-- Size of the data item to download -->
         <Size xmlns='syncml:metinf'>37214</Size>
      </Meta>
      <Target><LocURI>./antivirus_data</LocURI></Target>
      <Data>
         <!-- Base64-coded antivirus file -->
      </Data>
   </Item>
</Add>
```

Progress indicator will be displayed during the execution of the `Add` command and it will be scaled so that the total length of data to be downloaded is supposed to be 37214 bytes.

10.3 User interaction options

`Alert`'s MAY have optional User interaction parameters in the first `Item`. Optional parameters are represented as one text string inside the `Data` element. If the User interaction `Alert` does not have optional parameters, the first `Item` is empty. The optional parameter string conforms to the URL encoding format specified in [RFC2396].

The following example uses two optional parameters:

```
MAXDT=30&DR=1
```

The client MUST skip without error message all the optional parameters that it is not able to process.

The following optional parameters are currently defined.

10.3.1 MINDT (Minimum Display Time)

This parameter is a hint to the user agent of the minimum time that the user interaction should be displayed to the user. This can be important to guarantee that a notification message is readable.

MINDT parameter MUST have a value that can be evaluated as a positive, integer number. Value of MINDT is interpreted as notification display time to user in seconds.

Example:

```
<!-- Display this message for at least 10 seconds -->
<Item><Data>MINDT=10</Data></Item>
```

10.3.2 MAXDT (Maximum Display Time)

This parameter is a hint to the user agent for how long the client should wait for the user to execute the user interaction. If the user does not act within MAXDT time, the action is considered to be cancelled and a timeout status package or default response package is sent back to the server.

MAXDT parameter MUST have a value that can be evaluated as a positive, integer number. Value of MAXDT is interpreted as seconds to wait for user action.

Example:

```
<!-- Wait maximum 20 seconds for the user -->
<Item><Data>MAXDT=20</Data></Item>
```

10.3.3 DR (Default Response)

DR optional parameter specifies the initial state of the user interaction control widget. Other than setting the initial state of the user interaction control widget, DR has no other influence on the user interaction control widget. Interpretation for different user interaction types is the following:

- If the user interaction is Notification, this optional parameter is ignored.
- If the user interaction is a confirmation, 0 means that the reject user interface element is highlighted by default, 1 means that the accept user interface element is highlighted by default. Highlighted user interface element means that the "default" user interaction (like pressing Enter button) will select the highlighted user interface element. If the client user interface has no notion of highlighted user interface element, this parameter MAY be ignored.
- If the user interaction is user input, DR value specifies the original text in the text input user interface element. This text MUST conform to the optional parameter syntax rules.
- If the user interaction is single-choice, the DR value is the originally highlighted choice item; e.g. value between 1 and the number of items in the selection list.
- If the user interaction is a multi-choice, the DR value is a minus sign-separated list of originally highlighted values (for example: 2-3).

Examples:

```
<!-- Accept by default in a Confirmation action -->
<Item><Data>DR=1</Data></Item>
```

```
<!-- Default user entry of ‘John Doe’ in an user input action -->
<Item><Data>DR=John+Doe</Data></Item>
```

```
<!-- Default selection of item 3 in a single-choice action -->
<Item><Data>DR=3</Data></Item>
```

```
<!-- Default selection of item 2 and 3 in a multi-choice action -->
<Item><Data>DR=2-3</Data></Item>
```

10.3.4 MAXLEN (Maximum length of user input)

MAXLEN value is evaluated to a positive integer and determines the maximum number of characters that can be typed into the text input user interaction widget. The optional parameter MUST be ignored in all other kind of user interaction widget. If the specified maximum length of input string exceeds the capability of the client, the client MAY ignore the parameter.

Example:

```
<!-- Maximum string length is 30 -->
<Item><Data>MAXLEN=30</Data></Item>
```

10.3.5 IT (Input Type)

IT specifies what kind of characters is allowed in the text input user interaction widget. Based on this information a client with limited keyboard MAY display user interaction elements that allow easy input of characters not present on the keyboard. The optional parameter MUST be ignored in user interaction widgets other than text input. Allowed values:

IT=A - Alphanumeric input, client SHOULD allow input of all alphanumeric characters. This is the default behaviour.

IT=N - Numeric input, client SHOULD allow input of all numeric characters, decimal point and sign character.

IT=D - Date input, client SHOULD allow input of all numeric characters. User input is delivered to server in following text string format "DDMMYYYY", where;

- DD is day with possible leading zero.
- MM is month with possible leading zero.
- YYYY is year presented with four digits.

IT=T - Time input, client SHOULD allow input of all numeric characters. User input is delivered to server in following text string format "hhmmss", where;

- hh is hours with possible leading zero.
- mm is minutes with possible leading zero.
- ss is seconds with possible leading zero.

IT=P - Phone number input, client SHOULD allow input of all numeric characters, "+", "p", "w" and "s". "+" MUST be first if present in phone number.

IT=I - IP address input, client SHOULD allow input of all numeric characters. User input is delivered to server in following text string format "xxx.yyy.zzz.www"

Example:

```
<!-- Numeric text input -->
<Item><Data>IT=N</Data></Item>
```

`Status` message delivered to server as response

```
<Status>
   <MsgRef>1</MsgRef>
   <CmdRef>2</CmdRef>
   <Cmd>Alert</Cmd>
   <Data>200</Data> <!-- Successful, entered a number -->
   <Item>
      <Data>-1.23</Data>
   </Item>
</Status>
```

10.3.6 ET (Echo Type)

ET specifies how text input user interaction widget echoes the characters that the user types in. The optional parameter MUST be ignored in user interaction widgets other than text input. Allowed values:

ET=T - Text input. The client SHOULD allow the user to see the character the user typed into the text input user interaction widget. This is the default behaviour.

ET=P - Password input. The client SHOULD hide the character the user typed into the text input user interaction widget. One way of doing it MAY be writing an asterisk instead of the character itself.

Example:

```
<!-- Numeric text input -->
<Item><Data>ET=T</Data></Item>
```

11.Protocol examples

In this section several protocol scenarios will be demonstrated.

11.1 One-step protocol initiated by the server

In this section an example is presented in which a WAP connectivity context is added to the WAP settings. The user is asked to confirm whether the settings could be added.

11.1.1 Package 1: Initialization from client to server

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto> DM/1.3</VerProto>
    <SessionID>1</SessionID>
    <MsgID>1</MsgID>
    <Target>
      <LocURI>http://www.syncml.org/mgmt-server</LocURI>
    </Target>
    <Source>
      <LocURI>IMEI:493005100592800</LocURI>
    </Source>
    <Cred> <!-- Client credentials are mandatory if the transport layer is
    not providing authentication.-->
      <Meta>
        <Type xmlns='syncml:metinf'>syncml:auth-basic</Type>
        <Format xmlns='syncml:metinf'>b64</Format>
      </Meta>
      <Data>
        <!-- base64 formatting of userid:password -->
      </Data>
    </Cred>
    <Meta> <!-- Maximum message size for the client -->
      <MaxMsgSize xmlns='syncml:metinf'>5000</MaxMsgSize>
    </Meta>
  </SyncHdr>
  <SyncBody>
    <Alert>
      <CmdID>1</CmdID>
      <Data>1200</Data> <!-- Server-initiated session -->
    </Alert>
    <Replace>
      <CmdID>3</CmdID>
      <Item>
        <Source><LocURI>./DevInfo/DevId</LocURI></Source>
        <Meta>
          <Format xmlns='syncml:metinf'>chr</Format>
          <Type xmlns='syncml:metinf'>text/plain</Type>
        </Meta>
        <Data>IMEI:493005100592800</Data>
      </Item>
      <Item>
        <Source><LocURI>./DevInfo/Man</LocURI></Source>
        <Meta>
          <Format xmlns='syncml:metinf'>chr</Format>
          <Type xmlns='syncml:metinf'>text/plain</Type>
          </Meta>
          <Data>Device Factory, Inc.</Data>
        </Item>
        <Item>
          <Source><LocURI>./DevInfo/Mod</LocURI></Source>
          <Meta>
            <Format xmlns='syncml:metinf'>chr</Format>
            <Type xmlns='syncml:metinf'>text/plain</Type>
          </Meta>
          <Data>SmartPhone2000</Data>
        </Item>
        <Item>
          <Source><LocURI>./DevInfo/DmV</LocURI></Source>
          <Meta>
            <Format xmlns='syncml:metinf'>chr</Format>
            <Type xmlns='syncml:metinf'>text/plain</Type>
          </Meta>
          <Data>1.0.0.1</Data>
        </Item>
        <Item>
          <Source><LocURI>./DevInfo/Lang</LocURI></Source>
          <Meta>
            <Format xmlns='syncml:metinf'>chr</Format>
            <Type xmlns='syncml:metinf'>text/plain</Type>
          </Meta>
          <Data>en-US</Data>
        </Item>
      </Replace>
      <Final/>
  </SyncBody>
</SyncML>
```

11.1.2 Package 2: Initialization from server to client

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto> DM/1.3</VerProto>
    <SessionID>1</SessionID>
    <MsgID>1</MsgID>
    <Target>
      <LocURI>IMEI:493005100592800</LocURI>
    </Target>
    <Source>
      <LocURI>http://www.syncml.org/mgmt-server</LocURI>
    </Source>
    <Cred> <!-- Server credentials -->
      <Meta>
        <Type xmlns='syncml:metinf'>syncml:auth-basic</Type>
        <Format xmlns='syncml:metinf'>b64</Format>
      </Meta>
      <Data><!-- base64 formatting of userid:password --></Data>
    </Cred>
  </SyncHdr>
  <SyncBody>
    <Status>
```

```
        <MsgRef>1</MsgRef><CmdRef>0</CmdRef>
        <Cmd>SyncHdr</Cmd>
        <CmdID>6</CmdID>
        <TargetRef>http://www.syncml.org/mgmt-server</TargetRef>
        <SourceRef>IMEI:493005100592800</SourceRef>
        <!-- Authenticated for the session -->
        <Data>212</Data>
      </Status>
      <Status>
        <MsgRef>1</MsgRef><CmdRef>1</CmdRef>
        <CmdID>7</CmdID>
        <Cmd>Alert</Cmd>
        <Data>200</Data><!-- OK -->
      </Status>
      <Status>
        <MsgRef>1</MsgRef><CmdRef>3</CmdRef>
        <CmdID>8</CmdID>
        <Cmd>Replace</Cmd>
        <Data>200</Data><!-- OK -->
      </Status>
      <Sequence>
        <CmdID>1</CmdID>
        <Alert>
          <CmdID>2</CmdID>
          <Data>1101</Data> <!-- User confirmation required -->
          <Item></Item>
          <Item>
           <Data>Do you want to add the CNN access point?</Data>
          </Item>
        </Alert>
        <Replace>
          <CmdID>4</CmdID>
          <Meta>
            <Format xmlns='syncml:metinf'>b64</Format>
            <Type xmlns='syncml:metinf'>
              application/vnd.wap.connectivity-wbxml
            </Type>
          </Meta>
          <Item>
            <!-- CNN WAP settings object in the settings -->
            <Target>
              <LocURI>./settings/wap_settings/CNN</LocURI>
            </Target>
            <Data><!-- Base64-coded WAP connectivity document --></Data>
          </Item>
        </Replace>
      </Sequence>
      <Final/>
    </SyncBody>
</SyncML>
```

11.1.3 Package 3: Client response

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
    <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto> DM/1.3</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
      <Target>
         <LocURI>http://www.syncml.org/mgmt-server</LocURI>
      </Target>
      <Source>
         <LocURI>IMEI:493005100592800</LocURI>
      </Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <MsgRef>1</MsgRef>
         <CmdID>1</CmdID>
         <CmdRef>0</CmdRef>
         <Cmd>SyncHdr</Cmd>
         <!-- SyncHdr accepted -->
         <Data>212</Data>
      </Status>
      <Status>
         <MsgRef>1</MsgRef>
         <CmdID>2</CmdID>
         <CmdRef>1</CmdRef>
         <Cmd>Sequence</Cmd>
         <!-- Sequence executed correctly -->
         <Data>200</Data>
      </Status>
      <Status>
         <MsgRef>1</MsgRef><CmdRef>2</CmdRef>
         <CmdID>3</CmdID>
         <Cmd>Alert</Cmd>
         <!-- OK, the user confirmed the action -->
         <Data>200</Data>
      </Status>
      <Status>
         <MsgRef>1</MsgRef>
         <CmdRef>4</CmdRef>
         <CmdID>4</CmdID>
         <Cmd>Replace</Cmd>
         <TargetRef>./settings/wap_settings/CNN</TargetRef>
         <!-- OK, access point added -->
         <Data>200</Data>
      </Status>
      <Final/>
   </SyncBody>
</SyncML>
```

11.1.4 Package 4: Acknowledgement of client status

This package is now empty as no actions are sent and client does not continue the protocol.

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto> DM/1.3</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
      <Target>
         <LocURI>IMEI:493005100592800</LocURI>
      </Target>
      <Source>
         <LocURI>http://www.syncml.org/mgmt-server</LocURI>
      </Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <MsgRef>2</MsgRef>
         <CmdID>1</CmdID>
         <CmdRef>0</CmdRef>
         <Cmd>SyncHdr</Cmd>
         <Data>200</Data>
      </Status>
      <Final/>
   </SyncBody>
</SyncML>
```

11.2 Two-step protocol initiated by the server

Operator initiates a regular antivirus software update on PDA clients. The server checks the installed version in the first management section then updates the antivirus data in the second step.

11.2.1 Package 1: Initialization from client to server

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto> DM/1.3</VerProto>
      <SessionID>1</SessionID>
      <MsgID>1</MsgID>
      <Target>
         <LocURI>http://www.syncml.org/mgmt-server</LocURI>
      </Target>
      <Source>
         <LocURI>IMEI:493005100592800</LocURI>
      </Source>
      <Cred> <!-- Client credentials are optional -->
         <Meta>
            <Type xmlns='syncml:metinf'>syncml:auth-basic</Type>
            <Format xmlns='syncml:metinf'>b64</Format>
         </Meta>
         <Data><!-- base64 formatting of userid:password --></Data>
      </Cred>
      <Meta><!-- Maximum message size for the client -->
         <MaxMsgSize xmlns='syncml:metinf'>5000</MaxMsgSize>
      </Meta>
   </SyncHdr>
   <SyncBody>
      <Alert>
         <CmdID>1</CmdID>
         <Data>1200</Data> <!-- Server-initiated session -->
      </Alert>
      <Replace>
```

```
        <CmdID>3</CmdID>
        <Item>
          <Source><LocURI>./DevInfo/DevId</LocURI></Source>
          <Meta>
            <Format xmlns='syncml:metinf'>chr</Format>
            <Type xmlns='syncml:metinf'>text/plain</Type>
          </Meta>
          <Data>IMEI:493005100592800</Data>
        </Item>
        <Item>
          <Source><LocURI>./DevInfo/Man</LocURI></Source>
          <Meta>
            <Format xmlns='syncml:metinf'>chr</Format>
            <Type xmlns='syncml:metinf'>text/plain</Type>
          </Meta>
          <Data>Device Factory, Inc.</Data>
        </Item>
        <Item>
          <Source><LocURI>./DevInfo/Mod</LocURI></Source>
          <Meta>
            <Format xmlns='syncml:metinf'>chr</Format>
            <Type xmlns='syncml:metinf'>text/plain</Type>
          </Meta>
          <Data>SmartPhone2000</Data>
        </Item>
        <Item>
          <Source><LocURI>./DevInfo/DmV</LocURI></Source>
          <Meta>
            <Format xmlns='syncml:metinf'>chr</Format>
            <Type xmlns='syncml:metinf'>text/plain</Type>
          </Meta>
          <Data>1.0.0.1</Data>
        </Item>
        <Item>
          <Source><LocURI>./DevInfo/Lang</LocURI></Source>
          <Meta>
            <Format xmlns='syncml:metinf'>chr</Format>
            <Type xmlns='syncml:metinf'>text/plain</Type>
          </Meta>
          <Data>US-en</Data>
        </Item>
      </Replace>
      <Final/>
    </SyncBody>
</SyncML>
```

11.2.2 Package 2: Initialization from server to client

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
    <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto> DM/1.3</VerProto>
      <SessionID>1</SessionID>
      <MsgID>1</MsgID>
      <Target>
        <LocURI>IMEI:493005100592800</LocURI>
```

```
     </Target>
     <Source>
       <LocURI>http://www.syncml.org/mgmt-server</LocURI>
     </Source>
     <Cred> <!-- Server credentials -->
       <Meta>
         <Type xmlns='syncml:metinf'>syncml:auth-basic</Type>
         <Format xmlns='syncml:metinf'>b64</Format>
       </Meta>
       <Data><!-- base64 formatting of userid:password --></Data>
     </Cred>
   </SyncHdr>
   <SyncBody>
     <Status>
       <MsgRef>1</MsgRef><CmdRef>0</CmdRef>
       <Cmd>SyncHdr</Cmd>
       <CmdID>5</CmdID>
       <TargetRef>http://www.syncml.org/mgmt-server</TargetRef>
       <SourceRef>IMEI:493005100592800</SourceRef>
       <!-- Authenticated for the session -->
       <Data>212</Data>
     </Status>
     <Status>
       <MsgRef>1</MsgRef><CmdRef>1</CmdRef>
       <CmdID>6</CmdID>
       <Cmd>Alert</Cmd>
       <Data>200</Data><!-- OK -->
     </Status>
     <Status>
       <MsgRef>1</MsgRef><CmdRef>3</CmdRef>
       <CmdID>7</CmdID>
       <Cmd>Replace</Cmd>
       <Data>200</Data><!-- OK -->
     </Status>
     <Alert>
       <CmdID>2</CmdID>
       <Data>1100</Data> <!-- User displayable notification -->
       <Item></Item>
       <Item>
         <Data>Your antivirus software is being updated</Data>
       </Item>
     </Alert>
      <!-- Let's get the installed antivirus definition version number now -
->
     <Get>
       <CmdID>4</CmdID>
       <Item>
         <Target>
          <LocURI>./antivirus_data/version</LocURI>
         </Target>
       </Item>
     </Get>
     <Final/>
   </SyncBody>
</SyncML>
```

11.2.3 Package 3: Client response

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto> DM/1.3</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target>
      <LocURI>http://www.syncml.org/mgmt-server</LocURI>
    </Target>
    <Source>
      <LocURI>IMEI:493005100592800</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <MsgRef>1</MsgRef>
      <CmdID>1</CmdID>
      <Cmd>SyncHdr</Cmd>
      <Data>212</Data>
    </Status>
    <Status>
      <MsgRef>1</MsgRef>
      <CmdRef>2</CmdRef>
      <CmdID>2</CmdID>
      <Cmd>Alert</Cmd>
      <Data>200</Data><!-- User notification OK -->
    </Status>
    <Status>
      <MsgRef>1</MsgRef>
      <CmdRef>4</CmdRef>
      <CmdID>4</CmdID>
      <Cmd>Get</Cmd>
      <TargetRef>./antivirus_data/version</TargetRef>
      <Data>200</Data><!-- Get OK -->
    </Status>
    <!-- Results for the Get: antivirus version number -->
    <Results>
      <MsgRef>1</MsgRef><CmdRef>4</CmdRef>
      <CmdID>3</CmdID>
      <Item>
        <Source>
         <LocURI>./antivirus_data/version</LocURI>
        </Source>
        <Data>antivirus-inc/20010522b/5</Data>
      </Item>
    </Results>
    <Final/>
  </SyncBody>
</SyncML>
```

11.2.4 Package 4: Continue with management operations

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
```

```
      <VerDTD>1.2</VerDTD>
      <VerProto> DM/1.3</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
      <Target>
      <LocURI>IMEI:493005100592800</LocURI>
      </Target>
      <Source>
      <LocURI>http://www.syncml.org/mgmt-server</LocURI>
      </Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <MsgRef>2</MsgRef>
         <CmdID>1</CmdID>
         <Cmd>SyncHdr</Cmd>
         <Data>212</Data>
      </Status>
      <!-- Send now antivirus updates -->
      <Replace>
         <CmdID>2</CmdID>
         <Meta>
            <Format xmlns='syncml:metinf'>b64</Format>
            <Type xmlns='syncml:metinf'>
               application/antivirus-inc.virusdef
            </Type>
         </Meta>
         <Item>
            <Target>
              <LocURI>./antivirus_data</LocURI>
            </Target>
            <Data><!-- Base64-coded antivirus file --></Data>
         </Item>
      </Replace>
      <Final/>
   </SyncBody>
</SyncML>
```

11.2.5 Restart protocol iteration with Package 3: Client response

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto> DM/1.3</VerProto>
      <SessionID>1</SessionID>
      <MsgID>3</MsgID>
      <Target>
         <LocURI>http://www.syncml.org/mgmt-server</LocURI>
      </Target>
      <Source>
         <LocURI>IMEI:493005100592800</LocURI>
      </Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <MsgRef>2</MsgRef>
         <CmdID>1</CmdID>
         <Cmd>SyncHdr</Cmd>
         <Data>200</Data>
      </Status>
      <Status>
         <MsgRef>1</MsgRef>
         <CmdRef>2</CmdRef>
         <CmdID>2</CmdID>
         <Cmd>Replace</Cmd>
         <TargetRef>./antivirus_data</TargetRef>
         <!-- OK, antivirus update loaded -->
         <Data>200</Data>
      </Status>
      <Final/>
   </SyncBody>
</SyncML>
```

11.2.6 Package 4: Finish the protocol, no continuation

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto> DM/1.3</VerProto>
    <SessionID>1</SessionID>
    <MsgID>3</MsgID>
    <Target>
      <LocURI>IMEI:493005100592800</LocURI>
    </Target>
    <Source>
      <LocURI>http://www.syncml.org/mgmt-server</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <MsgRef>3</MsgRef>
      <CmdID>1</CmdID>
      <CmdRef>0</CmdRef>
      <Cmd>SyncHdr</Cmd>
      <Data>200</Data>
    </Status>
    <Final/>
  </SyncBody>
</SyncML>
```

12.Backward Compatibility

DM Servers MUST be compatible with all older minor and the same versions of the DM protocol. DM Clients MUST be compatible with the same version of the DM protocol and MAY be compatible with older versions of the DM protocol.

For example, a DM Server version 1.3 will have to be able to manage DM 1.1, DM 1.2 and DM 1.3 Clients. A DM Server version 1.2 will have to be able to manage DM 1.1 and DM 1.2 Clients.

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
N/A	N/A	No prior 1.3 version

A.2 Draft/Candidate Version 1.3 History

Document Identifier	Date	Sections	Description
Draft Versions OMA-TS-DM_Protocol-V1_3	15 Oct 2008	All	Baseline to v1.3 using OMA-TS-DM_Protocol-V1_2_1-20080617-A .
	09 Sep 2009	8.7	Applied OMA-DM-DM13-2009-0038-CR_DMProto_Alert_Code_Enhancement.
	28 Oct 2009	All	Applied OMA-DM-DM13-2009-0080R01-CR_Backward_CompatibilitiyOMA-DM-DM13-2009-0094R02-CR_Package1_Description OMA-DM-DM13-2009-0012-CR_Alert__Non_Visual___Bug__Fix.
	02 Nov 2009	All	Removed CR_Alert_Non_Visual_Bug_Fix (was not Agreed). Applied OMA-DM-DM13-2009-0106R01-CR_Protocol_cleanup
	10 Dec 2009	All	Applied OMA-DM-DM13-2009-0118R06-CR_RelativeURI_Addressing.
	28 Dec 2009	All	Applied OMA-DM-DM13-2009-0121R02-CR_Protocol_Pkg1_DevInfoDetail OMA-DM-DM13-2009-0127-CR_VerProto_Bug_Fix_for_DM_Protocol
	29 Dec 2009	All	Fixed clerical errors, seplling mistakes, etc.
	03 Feb 2010	B.1.1	Applied OMA-DM-DM13-2010-0013R01-CR_Protocol_SCR OMA-DM-DM13-2010-0021R01-CR_RelativeURI_Enhancement OMA-DM-DM13-2010-0033-CR_SCR_Entries_for_Protocol OMA-DM-DM13-2009-0134R02-CR_Protocol_Bug_Fixes
	11 Feb 2010	All	Editorial clean-up of formatting
	15 Mar 2010	All	Changed all text to UK. Reapplied OMA-DM-DM13-2010-0021R01-CR_RelativeURI_Enhancement.
	23 Apr 2010	All	Editorial cleanup.
	26 Apr 2010	T.O.C	Update of T.O.C (figures)
	04 May 2010	3.1, 10.2, 10.2.5, 10.3.5, 11.2.1, 11.2.4	3.1, 10.2 and 10.3.5: grammatical correction Double quotes changed to single quotes
	05 May 2010	All	Formatting of bullets Formatting of note Double quotes changed to single quotes Snippets font changed from Courrier to Courrier New to harmonize the snippets fonts
Candidate Version OMA-TS-DM_Protocol-V1_3	25 May 2010	N/A	Status changed to Candidate by TP Ref # OMA-TP-2010-0221-INP_DM_V1.3_ERP_and_ETR_for_Candidate_approval

Appendix B. Static Conformance Requirements (Normative)

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

B.1 SCR for DM Client

Item	Function	Reference	Status	Requirement
DM-PRO-C-001	Support of Session Setup Phase	8	M	
DM-PRO-C-002	Support of Session Abort	8.1	M	
DM-PRO-C-003	Support of Multiple Messages	6	O	
DM-PRO-C-004	Support of Large Object Handling. This is RECOMMENDED for clients.	7	O	DM-PRO-LO-C-001 AND DM-PRO-LO-C-002 AND DM-PRO-LO-C-003 AND DM-PRO-LO-C-004 AND DM-PRO-LO-C-005 AND DM-PRO-LO-C-006 AND DM-PRO-LO-C-008 AND DM-PRO-LO-C-009 AND DM-PRO-LO-C-010
DM-PRO-C-005	Support of Management Phase	8	M	
DM-PRO-C-006	Support for executing Management Commands	8	M	
DM-PRO-C-007	Executing User Interaction Commands	10	O	(DM-PRO-UI-C-001 OR DM-PRO-UI-C-002 OR DM-PRO-UI-C-003 OR DM-PRO-UI-C-004 OR DM-PRO-UI-C-005) AND DM-PRO-UI-C-006
DM-PRO-C-008	Support for sending Status and Result after receiving Management Operations	8.4	M	
DM-PRO-C-009	Support for standard SyncML command Format and Status and Result reporting	8	M	
DM-PRO-C-010	Support for sending asynchronous data via client initiated Alerts	8.3	O	
DM-PRO-C-011	Sending Generic Alert	8.7	O	DM-PRO-GAlert-C-001 OR DM-PRO-GAlert-C-002
DM-PRO-C-012	Support Relative URI Addressing Mechanism	5.2	O	

B.1.1 SCR for DM Session Setup Phase

Item	Function	Reference	Status	Requirement
DM-PRO-Session-C-001	Support Server Notification	8	O	
DM-PRO-Session-C-002	Sending Client Initiation and Device Info (Package #1) including Final element	8.3	M	
DM-PRO-Session-C-003	Sending Alert in the management session with data being either Client-Initiated 1201 or Server-Initiated 1200	8.3	O	
DM-PRO-Session-C-	Sending Device Info in Replace	8.3	M	

Item	Function	Reference	Status	Requirement
004	Command in Package #1			

B.1.2 SCR for Session Abort

Item	Function	Reference	Status	Requirement
DM-PRO-Abort-C-001	Sending Session Abort Alert	8.1	O	DM-PRO-Abort-C-004 AND DM-PRO-Abort-C-005
DM-PRO-Abort-C-002	Receiving Session Abort Alert	8.1	M	
DM-PRO-Abort-C-003	Session Abort message includes Status and Results of executed commands	8.1.2	O	DM-PRO-Abort-C-001
DM-PRO-Abort-C-004	Include Final in Message	8.1.2	O	
DM-PRO-Abort-C-005	Sender of Abort discards the response if response is received	8.1.1	O	

B.1.3 SCR for Multiple Messages

Item	Function	Reference	Status	Requirement
DM-PRO-Mul-C-001	Last message within multiple messages must contain Final	6.2	M	
DM-PRO-Mul-C-002	If message that is not the last one within Multiple Messages then the Next Message or Abort Alert must be sent	6.2	M	

B.1.4 SCR for Large Object

Item	Function	Reference	Status	Requirement
DM-PRO-LO-C-001	Support for Large Object	7	O	DM-PRO-LO-C-002 AND DM-PRO-LO-C-003 AND DM-PRO-LO-C-004 AND DM-PRO-LO-C-005 AND DM-PRO-LO-C-006 AND DM-PRO-LO-C-008 AND DM-PRO-LO-C-009 AND DM-PRO-LO-C-010
DM-PRO-LO-C-002	Respond with Status 213 when a data chunk that is not the last one is received	7	O	
DM-PRO-LO-C-003	Management Commands inside Large Object are handled as Atomic	7	O	
DM-PRO-LO-C-004	While sending data chunks all chunks except the last one must include “MoreData”	7	O	
DM-PRO-LO-C-005	Data chunks must be sent in contiguous order without any new commands	7	O	
DM-PRO-LO-C-006	Data that fits into a single message	7	O	

Item	Function	Reference	Status	Requirement
	must be sent in a single message			
DM-PRO-LO-C-007	Sending MaxObjSize to indicate size limitations for Package	7	O	DM-PRO-C-004
DM-PRO-LO-C-008	Never create packages bigger than the size the Server indicated in MaxObjSize	7	O	
DM-PRO-LO-C-009	Include Size in first data chunk	7	O	
DM-PRO-LO-C-010	Compare actual size and the Size value and report if not equal	7	O	
DM-PRO-LO-C-011	Indicate support for Large Object in DevDetail	7	M	

B.1.5 SCR for User Interaction Commands

Item	Function	Reference	Status	Requirement
DM-PRO-UI-C-001	Executing Display Alert	10.2.1	O	
DM-PRO-UI-C-002	Executing Confirm or Reject Alert	10.2.2	O	
DM-PRO-UI-C-003	Executing Text Input Alert	10.2.3	O	
DM-PRO-UI-C-004	Executing Single Choice Alert	10.2.4	O	
DM-PRO-UI-C-005	Executing Multiple Choice Alert	10.2.4	O	
DM-PRO-UI-C-006	Order of the Items MUST be used in the same order as in the DM message	10.2	O	

B.1.6 SCR for Generic Alert

Item	Function	Reference	Status	Requirement
DM-PRO-GAlert-C-001	The Generic Alert has a relation to a Management Object	8.7	O	DM-PRO-GAlert-C-003
DM-PRO-GAlert-C-002	The Generic Alert does not have a relation to a Management Object	8.7	O	
DM-PRO-GAlert-C-003	LocURI must reference the address to the corresponding Management Object	8.7	O	
DM-PRO-GAlert-C-004	Support for Correlator	8.7.1.10	O	DM-PRO-C-011
DM-PRO-GAlert-C-005	Type must be included and it is RECOMMENDED to include URN or registered MIME-type as Type	8.7.1.6	O	DM-PRO-C-011
DM-PRO-GAlert-C-006	Support for importance level, Mark	8.7.1.8	O	DM-PRO-C-011

B.2 SCR for DM Server

Item	Function	Reference	Status	Requirement
DM-PRO-S-001	Support of Session Setup Phase	8	M	
DM-PRO-S-002	Support of Session Abort	8.1	M	
DM-PRO-S-003	Support of Multiple Messages	6	M	
DM-PRO-S-004	Support of Large Object Handling	7	M	
DM-PRO-S-005	Support of Management Phase	8	M	

Item	Function	Reference	Status	Requirement
DM-PRO-S-006	Support for sending Management Commands	8	M	
DM-PRO-S-007	Sending User Interaction Commands	10	O	(DM-PRO-UI-S-001 OR DM-PRO-UI-S-002 OR DM-PRO-UI-S-003 OR DM-PRO-UI-S-004 OR DM-PRO-UI-S-005) AND DM-PRO-UI-S-006
DM-PRO-S-008	Support for sending Status and Results on Client Commands and Alerts	8	M	
DM-PRO-S-009	Support of Generic Alert	8.7	M	
DM-PRO-S-010	Support application layer authentication	9	M	
DM-PRO-S-011	Support Relative URI Addressing Mechanism	5.2	O	

B.2.1 SCR for DM Session Setup Phase

Item	Function	Reference	Status	Requirement
DM-PRO-Session-S-001	Support Server Notification	8	O	SCR-DM-NOTI-S-001
DM-PRO-Session-S-002	Support of receiving initiation message from client (Package #1), perform authentication and send initiation (Package #2)	8.4	M	

B.2.2 SCR for Session Abort

Item	Function	Reference	Status	Requirement
DM-PRO-Abort-S-001	Sending Session Abort Alert	8.1	O	DM-PRO-Abort-S-004 AND DM-PRO-Abort-S-005
DM-PRO-Abort-S-002	Receiving Session Abort Alert	8.1	M	
DM-PRO-Abort-S-003	Session Abort message includes Status and Results of executed commands	8.1.2	O	DM-PRO-Abort-S-001
DM-PRO-Abort-S-004	Include Final in Message	8.1.2	O	
DM-PRO-Abort-S-005	Sender of Abort must discard the response if response is received	8.1.1	O	

B.2.3 SCR for Multiple Messages

Item	Function	Reference	Status	Requirement
DM-PRO-Mul-S-001	Last message within multiple messages must contain Final	6.2	M	
DM-PRO-Mul-S-002	If message that is not the last one within Multiple Messages then the Next Message or Abort Alert must be sent	6.2	M	

B.2.4 SCR for Large Object

Item	Function	Reference	Status	Requirement
DM-PRO-LO-S-001	Respond with Status 213 when a data chunk that is not the last one is received	7	M	
DM-PRO-LO-S-002	Management Commands inside Large Object are handled as Atomic	7	M	
DM-PRO-LO-S-003	While sending data chunks all chunks except the last one must include “MoreData”	7	M	
DM-PRO-LO-S-004	Data chunks must be sent in contiguous order without any new commands	7	M	
DM-PRO-LO-S-005	Data that fits into a single message must be sent in a single message	7	M	
DM-PRO-LO-S-006	Sending MaxObjSize to indicate size limitations for Package	7	O	
DM-PRO-LO-S-007	Never create packages bigger than the size the Client indicated in MaxObjSize	7	M	
DM-PRO-LO-S-008	Include Size in first data chunk	7	M	
DM-PRO-LO-S-009	Compare actual size and the Size value and report if not equal	7	M	

B.2.5 SCR for User Interaction Commands

Item	Function	Reference	Status	Requirement
DM-PRO-UI-S-001	Sending Display Alert	10.2.1	O	
DM-PRO-UI-S-002	Sending Confirm or Reject Alert	10.2.2	O	
DM-PRO-UI-S-003	Sending Text Input Alert	10.2.3	O	
DM-PRO-UI-S-004	Sending Single Choice Alert	10.2.4	O	
DM-PRO-UI-S-005	Sending Multiple Choice Alert	10.2.4	O	
DM-PRO-UI-S-006	Order of the Items MUST be followed in the DM message	10.2	O	

B.2.6 SCR for Generic Alert

Item	Function	Reference	Status	Requirement
DM-PRO-GAlert-S-001	Support for receiving, parsing and send Status back to client	8.7	M	
DM-PRO-GAlert-S-002	Perform action from the data content in the Generic Alert	8.7	O	

Appendix C. Protocol Values (Normative)

VerProto Codes	Description
DM/1.3	Indicates that this DM message uses the OMA Device Management Protocol defined by the Open Mobile Alliance.