DS Protocol

Approved Version 1.2.1 – 10 Aug 2007

Open Mobile Alliance

OMA-TS-DS_Protocol-V1_2_1-20070810-A

Contents

Figures

Tables

1. Scope

The SyncML Initiative, Ltd. was a not-for-profit corporation formed by a group of companies who co-operated to produce an open specification for data synchronization and device management. Prior to SyncML, data synchronization and device management had been based on a set of different, proprietary protocols, each functioning only with a very limited number of devices, systems and data types. These non-interoperable technologies have complicated the tasks of users, manufacturers, service providers, and developers. Further, a proliferation of different, proprietary data synchronization and device management protocols has placed barriers to the extended use of mobile devices, has restricted data access and delivery and limited the mobility of the users.

The SyncML Initiative merged with the Open Mobile Alliance in November 2002. The SyncML legacy specifications were converted to the OMA format with the 1.1.2 versions of OMA SyncML Common, OMA Data Synchronization and OMA Device Management in May 2002. The relationship between these documents which had been created during the SyncML Initiative has been preserved and is depicted in **Figure 1 OMA SyncML Specification Structure and Relationships**.

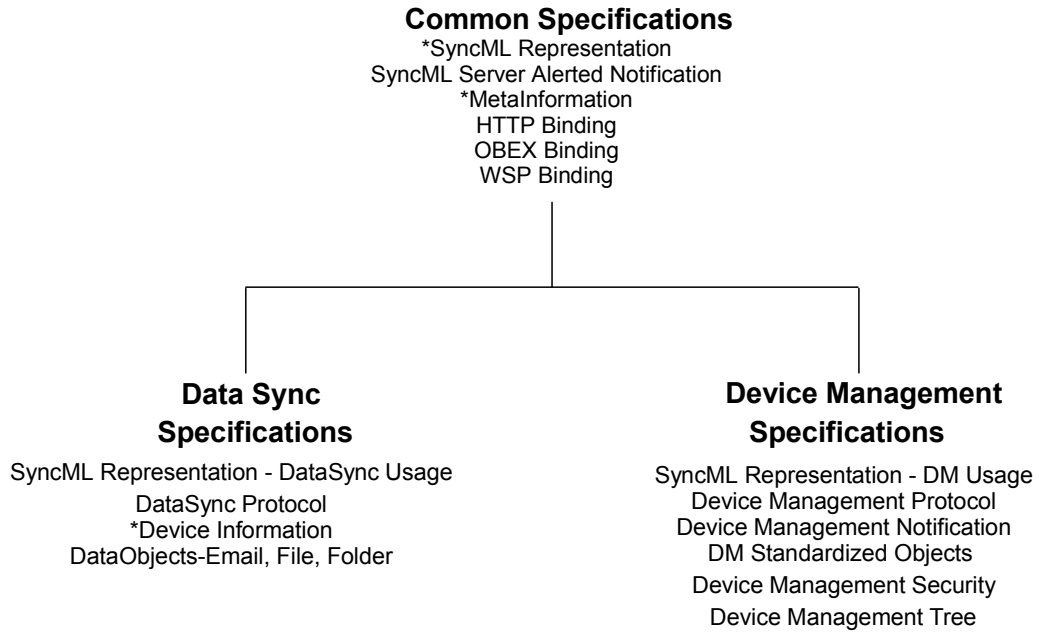

Figure 1 OMA SyncML Specification Structure and Relationships

The OMA SyncML Common Specifications Enabler Release includes the following documents:

- SyncML Representation: The XML-based representation protocol which specifies the common XML syntax and semantics used by all SyncML protocols. This defines the superset of the DS and DM representation protocols. (* includes DTD).
- The transport bindings: HTTP, OBEX, WSP. These specify the features REQUIRED for each transport to send and receive DS and DM protocol messages.
- The Meta Information associated with a SyncML command or data item or collection used by either DS or DM (* includes DTD)

- SyncML Server Alerted Notification: The logical structure and format of the notification messages used by all SyncML server alerted notifications, for both DS and DM.

The OMA DS (Data Synchronization) Specifications Enabler Release includes the following documents:

- SyncML Representation DataSync Usage: The subset of the Common Specifications SyncML Representation Specification necessary to define the Data Synchronization commands and protocol, with examples and commentary specific to DS.
- DataSync Protocol: Specifies how SyncML Common messages conforming to the DTD are exchanged in order to allow an OMA DS client and server to exchange additions, deletions, updates and other status information.
- Device Information: Used to exchange device specific information, including hardware, firmware, software levels, available memory, and local databases supported. (* Includes DTD)
- Data Objects: Email, File, Folder: Each object is identified by a unique MIME media type (eg. **application/vnd.omads-email)**. The objects are either represented by or encapsulated in a mark-up language defined by xml. Meta or state data is included in the representation (eg. Read/Unread, Creation Date, Last Modified Date).

Although the SyncML Common specification defines transport bindings that specify how to use a particular transport to exchange messages and responses, the SyncML Common representation, synchronization and device management protocols are transport-independent. Each package in these protocols is completely self-contained, and could in principle be carried by any transport. The initial bindings specified are HTTP, WSP and OBEX, but there is no reason why SyncML Common could not be implemented using email or message queues, to list only two alternatives. Because the SyncML Common messages are self-contained, multiple transports could be used without either the server or client devices having to be aware of the network topology. Thus, a short-range OBEX connection could be used for local connectivity, with the messages being passed on via HTTP to an Internet-hosted synchronization server.

To reduce the data size, a binary coding of SyncML Common based on the WAP Forum's WBXML is defined. Messages may also be passed in clear text if desired. In this and other ways SyncML Common addresses the bandwidth and resource limitations imposed by mobile devices.

SyncML Common is both data type and data store independent. SyncML Common can carry any data type which can be represented as a MIME object. To promote interoperability between different implementations of OMA Data Synchronization, the specification includes the representation formats used for common PIM data.

This document specifies the message flows between data synchronization client and server in order to ensure an inter-operable solution across all devices.

2. References

2.1 Normative References

[DEVINF] "OMA DS Device Information", Open Mobile Alliance™, OMA-TS-DS-DevInf-V1_2, URL:http:www.openmobilealliance.org.

[DSREPU] "SyncML Representation Protocol, Data Synchronization Usage", Open Mobile Alliance™, OMA-TS-DS_DataSyncRep-V1_2, URL:http:www.openmobilealliance.org..

[IMCVCAL] "vCalendar – The electronic calendaring and scheduling exchange format – Version 1.0", URL:http://www.imc.org/pdi/vcal-10.doc

[IMEI] "Digital cellular telecommunications system (Phase 2+) (GSM); Universal Mobile Telecommunications System (UMTS); Numbering, addressing and identification" (3G TS 23.003 Version 3.4.0 Release 1999), URL:http://webapp.etsi.org/action/PU/20000523/ts_123003v030400p.pdf

[META] "SyncML Meta Information protocol specification", Open Mobile Alliance™, OMA-SyncML-MetaInformation-V1_2, URL:http:www.openmobilealliance.org

[REPPRO] "SyncML Representation Protocol", Open Mobile Alliance™, OMA-TS-SyncML-RepPro-V1_2, URL:http:www.openmobilealliance.org

[RFC1321] "The MD5 Message-Digest Algorithm", URL:http://www.ietf.org/rfc/rfc1321.txt

[RFC2045] "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed & N. Borenstein, November 1966, URL:http://www.ietf.org/rfc/rfc2045.txt

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

[RFC2234] "Augmented BNF for Syntax Specifications: ABNF". D. Crocker, Ed., P. Overell. November 1997. URL:http://www.ietf.org/rfc/rfc2234.txt

[RFC2396] "Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, et al., August 1998, URL:http://www.ietf.org/rfc/rfc2396.txt

[SAN] "SyncML Server Alerted Notification", Open Mobile AllianceTM , OMA-SyncML-SAN-V1_2 , URL:http://www.openmobilealliance.org

[WBXML] "WAP Binary XML Content Format Specification", WAP Forum™, WAP-154-WBXML, URL:http://www.openmobilealliance.org

[WSPCTC] "WSP Content Type Codes" Open Mobile AllianceTM,

http://www.openmobilealliance.org/tech/omna/omna-wsp-content-type.htm

[XML] "Extensible Markup Language (XML) 1.0", World Wide Web Consortium Recommendation, URL:http://www.w3.org/TR/REC-xml

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 SyncML DTD or XML snippets is specified in this `typeface`.

3.2 Definitions

Application	An OMA DS application that supports the OMA DS protocol. The application can either be the originator or recipient of the SyncML protocol commands. The application can act as an OMA DS client or an OMA DS server.
Capabilities exchange	The OMA DS capability that allows a client and server to exchange what device, user and application features they each support.
Client	An OMA DS Client refers to the protocol role when the application issues SyncML "request" messages. For example in data synchronization, the Sync SyncML Command in a SyncML Message.
Client Modification	A modification of an item, which occurs in a client database before the modification is synchronized to the server database.
Command	A SyncML Command is a protocol primitive. Each SyncML Command specifies to a recipient an individual operation that is to be performed. For example, the SyncML Commands supported by this specification include Add, Alert, Atomic, Copy, Delete, Exec, Get, Map, Replace, Search, Sequence and Sync.
Data	A unit of information exchange, encoded for transmission over a network.
Data collection	A data element, which acts as a container of other data elements, (e.g., {c {{i1, data1}, ... {in, datan}}}). In OMA DS, data collections are synchronized with each other. See data element.
data element	A piece of data and an associated identifier for the data, (e.g., {i, data}).
Data element equivalence	When two data elements are synchronized. The exact semantics is defined by a given data synchronization model.
Data exchange	The act of sending, requesting or receiving a set of data elements.
Data format	The encoding used to format a data type. For example, characters or integers or character encoded binary data.
Data type	The schema used to represent a data object (e.g., text/calendar MIME content type for an iCalendar representation of calendar information or text/directory MIME content type for a vCard representation of contact information).
Data synchronization	The act of establishing an equivalence between two data collections, where each data element in one item maps to a data item in the other, and their data is equivalent.
Data synchronization protocol	The well-defined specification of the "handshaking" or workflow required to accomplish synchronization of data elements on an originator and recipient data collection. The OMA DS specification forms the basis for specifying an open data synchronization protocol.
GUID (Global Unique Identifier)	A number assigned to an object in a database. GUID values are never reused. Note that in practice, numbers do not have to be unique forever, they MUST only be unique as long as they exist in some mapping table.

LUID (Locally Unique Identifier)	A number assigned to an object in a database. LUID values are only unique locally, i.e., to a particular OMA DS client database, but MAY be present on other OMA DS client databases. In this protocol, the OMA DS client device assigns to each object a locally unique, non-reusable identifier, or LUID. They are unique per device and per application.
Message	A SyncML Message is the primary contents of a SyncML Package. It contains the SyncML Commands, as well as the related data and meta-information. The SyncML Message is an XML document.
Operation	A SyncML Operation refers to the conceptual transaction achieved by the SyncML Commands specified by a SyncML Package. For example in the case of data synchronization, "synchronize my personal address book with a public address book".
Originator	The network device that creates a SyncML request.
Package	A SyncML Package is the complete set of commands and related data elements that are transferred between an originator and a recipient. The SyncML package can consist of one or more SyncML Messages.
Parser	Refers to an XML parser. An XML parser is not absolutely required to support SyncML. However, an OMA DS implementation that integrates an XML parser may be easier to enhance. This document assumes that the reader has some familiarity with XML syntax and terminology.
Recipient	The network device that receives a SyncML request, processes the request and sends any resultant SyncML response.
Representation protocol	A well-defined format for exchanging a particular form of information. SyncML is a representation protocol for conveying data synchronization and device management operations.
Request	A message or a command sent from a device to another.
Server	An OMA DS Server refers to the protocol role when an application issues SyncML "response" messages. For example in the case of data synchronization, a Results Command in a SyncML Message.
Server alerted notification	The general term for Server Alerter Synchronization
Server modification	A modification of an item, which occurs in the server database before the modification is synchronized to the client database.
Slow synchronization	When a data set is synchronized for the first time, or state relating to the synchronization has been lost, the whole data set MUST be copied from one device to the other. Since this can be a time-consuming operation, this is known as slow synchronization.
Synchronization anchor	A string representing a synchronization event. The format of the string will typically be either a sequence number or an ISO 8601-formatted extended representation, basic format date/time stamp.
Synchronization data	Refers to the data elements within a SyncML Command. In a general reference, can also refer to the sum of the data elements within a SyncML Message or SyncML Package.
Synchronization engine	The portion of an OMA DS server that can analyze a data set and modifications to that data set made by both OMA DS server and OMA DS client. The synchronization engine will implement policies to enable the detection and resolution of conflicting changes.
SyncML request message	An initial SyncML Message that is sent by an originator to a recipient network device.
SyncML response message	A reply SyncML Message that is sent by a recipient of a SyncML Request back to the originator of the SyncML Request.
Temporary GUID	A temporary number assigned by the server to an object in a database (See also GUID.). Temporary GUID values are valid till the map operation for the items, with which the temporary GUIDs are associated, has been received from the client. After that the temporary GUID can be erased.

3.3 Abbreviations

ABNF Augmented Backus-Naur Form [RFC2234]

DTD Document Type Definition

GUID Global Unique Identifier

HTTP HyperText Transfer Protocol

IMEI International Mobile Equipment Identifier

LUID Local Unique Identifier

MD5 Message Digest algorithm version 5 [RFC1321]

MSC Message Sequence Chart

MSG Message

OBEX Object Exchange protocol

URI Uniform Resource Identifier [RFC2396]

URL Uniform Resource Locator [RFC2396]

WAP Wireless Application Protocol

WSP Wireless Session Protocol

XML Extensible Markup Language

4. Introduction

This specification defines a synchronization protocol between an OMA Data Synchronization client and server in the form of message sequence charts. It specifies how to use the SyncML Representation protocol so that interoperating client and server solutions are accomplished.

5. OMA DS Introduction

The purpose of this specification is to define a synchronization protocol using the SyncML Representation protocol [REPPRO]. This protocol is called the OMA Data Sync Protocol (OMA DS). This specification defines the protocol for different sync procedures, which can occur between a synchronizationclient and a synchronizationserver, in the form of message sequence charts (MSC's). The specification covers the most useful and common synchronization cases (Chapters 8 - 12).

5.1 SyncML Framework

This specification can be implemented by using the SyncML interface from the SyncML Framework (See Figure 2). Not all the features of the SyncML Interface need to be implemented to comply with this specification.

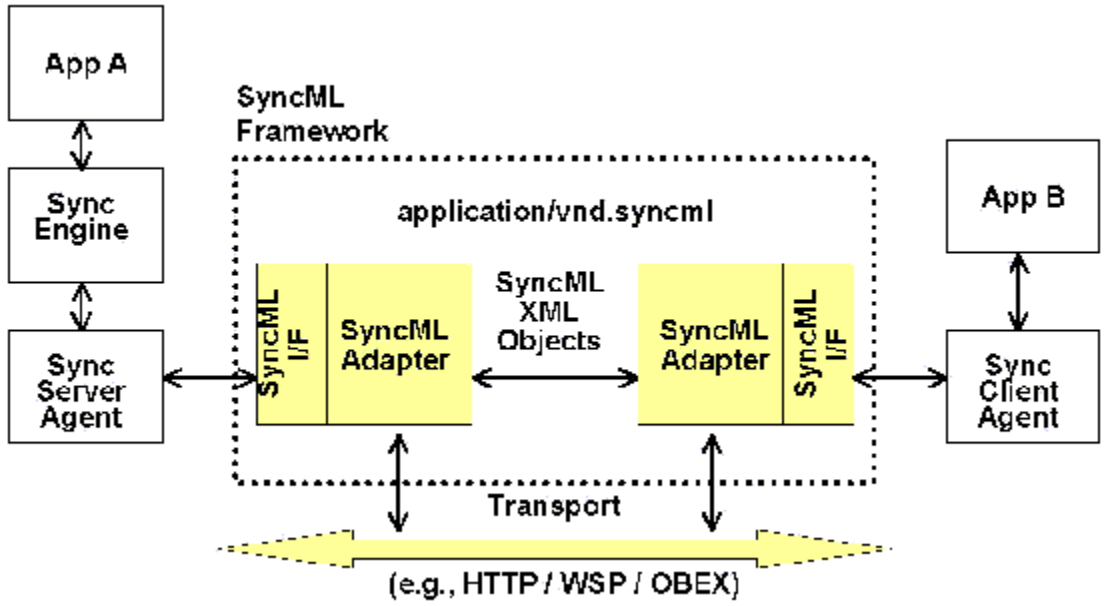

Figure 2 SyncML Framework

The application "A" depicts a networked service that provides data synchronization service for other applications, in this case application "B", on some networked device. The service and device are connected over some common network transport, such as HTTP.

In the figure above, the 'Sync Engine' functionality is completely placed onto the OMA DS server side even if some OMA DS client implementations can in practice provide some sync engine functionality, too. The 'Sync Server Agent' and the 'Sync Client Agent' use the protocol defined in this specification and the representation protocol [DSREPU] offered by the SyncML interface ('SyncML I/F') to communicate with each other.

5.2 Device Roles

Figure 3 depicts a synchronization example in which a mobile phone acts as an OMA DS client and a server acts as an OMA DS server. The client sends SyncML message including the data modifications made in the client to the server. The server synchronizes the data (including possible additions, replaces, and deletions) within the SyncML messages with data stored in the server. After that, the server returns its modifications back to the client.

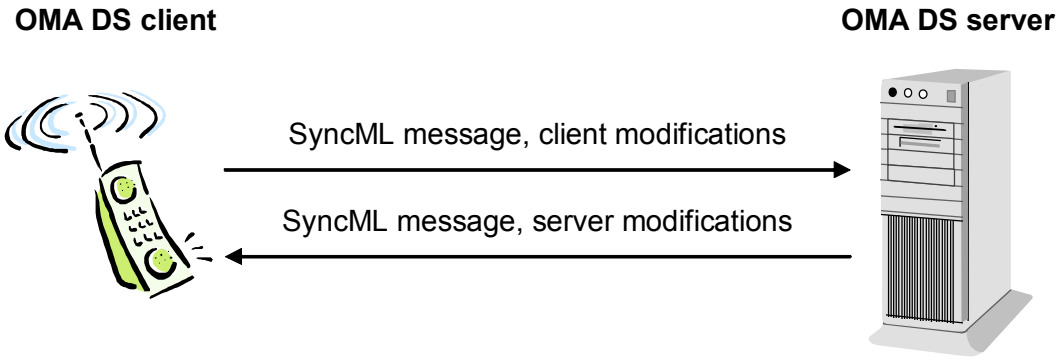

Figure 3 Synchronization Example with Mobile Phone and Server

The example presented the figure above is very simple. However, this example describes the roles of the devices in this specification. That is:

OMA DS Client – This is the device that contains a sync client agent and that sends first its modifications to the server. The client MUST also be able to receive responses from the OMA DS server. Although the OMA DS client has always the role to send its modifications first, in some cases the server can have a role to initiate synchronization. The OMA DS client is typically a mobile phone, PC, or PDA device.

OMA DS Server – This is the device, which contains a sync server agent and sync engine, and which usually waits that the OMA DS client starts synchronization and sends the clients modification to the server. The server is responsible for processing the sync analysis when it has received the client modifications. In addition, it can be able to initiate synchronization if unsolicited commands from the server to the client are supported on the transport protocol level. Typically, the OMA DS server is a server device or PC.

5.3 Sync Types

This specification defines seven different sync types. These are introduced in Table 1.

Table 1 OMA DS Sync Types

Sync Scenario	Description	Reference
Two-way sync	A normal sync type in which the client and the server exchange information about modified data in these devices. The client sends the modifications first.	Chapter 9
Slow sync	A form of two-way sync in which all items are compared with each other on a field-by-field basis. In practice, this means that the client sends all its data from a database to the server and the server does the sync analysis (field-by-field) for this data and the data in the server.	Chapter 9.5
One-way sync from client only	A sync type in which the client sends its modifications to the server but the server does not send its modifications back to the client.	Chapter 10
Refresh sync from client only	A sync type in which the client sends all its data from a database to the server (i.e., exports). The server is expected to replace all data in the target database with the data sent by the client.	Chapter 10.3
One-way sync from server only	A sync type in which the client gets all modifications from the server but the client does not send its modifications to the server.	Chapter 11

Refresh sync from server only	A sync type in which the server sends all its data from a database to the client. The client is expected to replace all data in the target database with the data sent by the server.	Chapter 11.5
Server Alerted Sync	A sync alert type, which provides the means for a server to alert the client to perform synchronization. When the server alerts the client, it also tells it which type of synchronization to initiate.	Chapter 12

6. Protocol Fundamentals

In this chapter, the common features and requirements for all sync types are defined.

6.1 Change Log Information

This protocol requires that devices (the client and server) are able to keep tracks of changes that have happened between synchronizations. I.e., they are responsible for maintaining the change log information about the modifications associated with data items of a database. The types of the modifications can be e.g., replace, addition, and deletion. This protocol does not specify in which format this change log information is maintained inside devices. However, when synchronization is started, the devices MUST be able to specify, which data items have changed. To specify the changed data items, the unique identifiers are used (See also Chapter 6.3). To indicate the type of a modification, the different operations (e.g., Replace) are used.

6.1.1 Multiple devices

If a device synchronizes with multiple devices, the change log information MUST be able to indicate all modifications related to a previous synchronization with each device.

6.2 Usage of Sync Anchors

6.2.1 Sync Anchors for Databases

To enable sanity checks of synchronization, this protocol uses sync anchors (See Definitions) associated with databases (e.g., a calendar database). There are two sync anchors, Last and Next (See [META]), which are always sent at the initialization of sync. The Last sync anchor describes the last event (e.g., time) when the database was synchronized from the point of sending device and the Next sync anchor describes the current event of sync from the point of sending device. Thus, both the client and the server send their own anchors to each other. The sync anchors are sent within the Meta information of an Alert operation by using the Meta Information DTD as defined by the SyncML Common specifications. The receiving device MUST echo the Next sync anchor back to the transmitting device in the Status for the Alert command (Data of the Item element inside Status).

The utilization of sync anchors is implementation specific but in order to enable the utilization, the Next sync anchor of another device needs to be stored until the next synchronization. The server MUST store the Next sync anchor sent by the client to enable this utilization.

If the device stores the Next sync anchor, it is able to compare during the next synchronization whether the sync anchor is the same as the Last sync anchor sent by another device. If they are matching, the device is able to conclude that no failures have happened since last sync. If they are not matching, the device can request a special action from another device (e.g., slow sync).

The stored sync anchors MUST NOT be updated before the synchronization session is finished.

The synchronization session is finished after a device is not going to send and is not expecting to receive any SyncML messages from other device, and the synchronization was successful on the Sync command level (i.e. no other than 200-class statuses has been returned for Sync commands). Also the transport level (directly under SyncML level) communication has to be properly ended before synchronization can be seen as finished. If the communication between synchronizing devices is not ended properly according to transport level specification, devices MUST NOT update their sync anchors. However, if the interrupted session is to be resumed then the value of the ‘Next’ anchor MUST be updated (See Section 6.13.2).

6.2.1.1 Example of Database Sync Anchor Usage

In this example, a sync client and server synchronize twice (sync sessions #1 and #2) with each other. After the sync session #1, the persistent memory of the sync client is reset. Because of that, the database anchors do not match at the sync session #2, the sync server notifies this, and it initiates the slow sync with the client.

The sync session #1 is started at 10:10:10 AM on the 10th of October 2001. The previous synchronization (before the sync session #1) was started at 09:09:09 AM on the 9th of September 2001. At this synchronization session, the slow sync is not initiated because the sync anchors match. I.e., the sync server has the sync event (09:09:09 AM on the 9th of September, 2001).

The sync session #2 is started at 11:11:11 AM on the 11th of November 2001. Because the memory of the sync client was reset after the sync session #1, the sync server initiates the slow sync.

In the figure below, both the sync sessions are depicted. Only the initialization phases and the client sync anchors are shown in the figure.

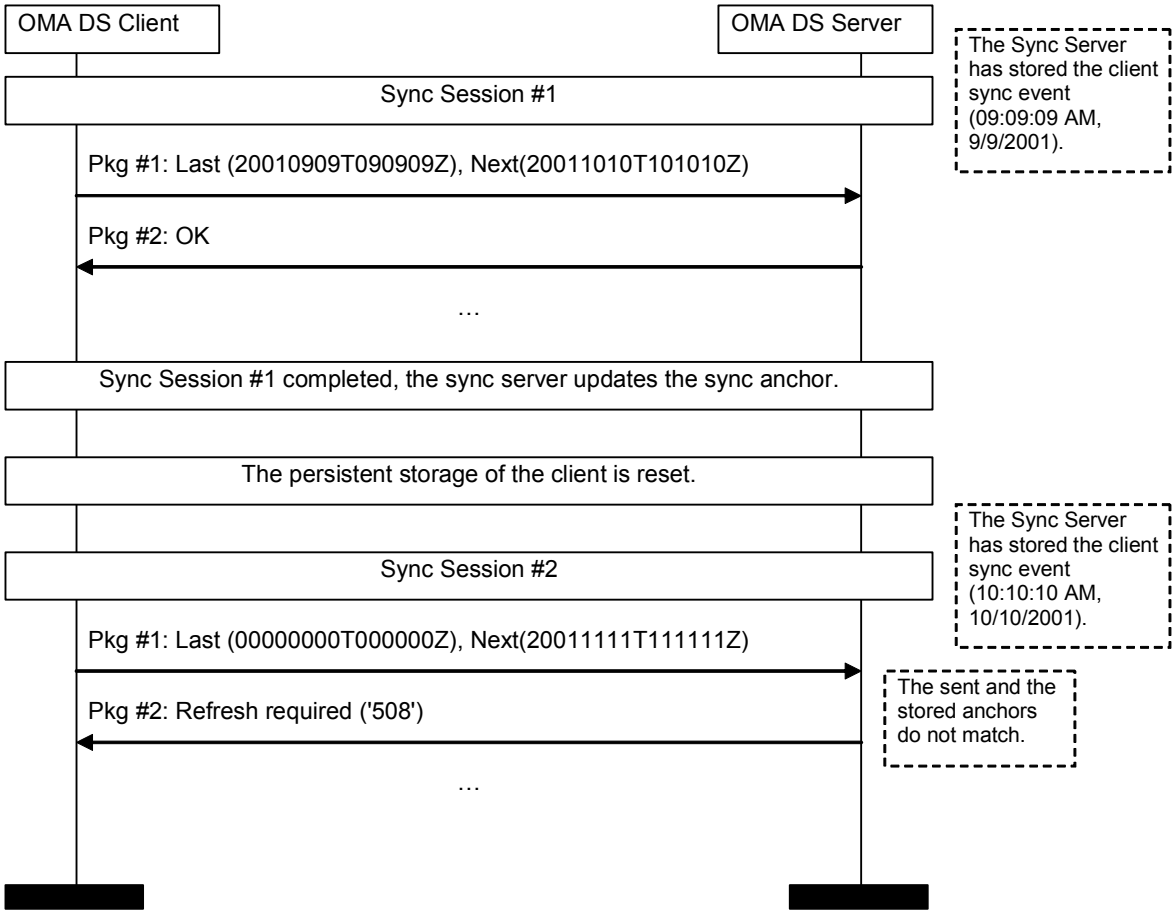

Figure 4 Example of Sync Anchor Usage

6.2.2 Sync Anchors for Data Items

This protocol does not specify the functionality to transfer the sync anchors associated with individual data items. If this functionality is desired, it MUST be provided inside the data items if it is included. An example is the Sequence Number property of vCalendar, the electronic calendaring and scheduling exchange format [IMCVCAL].

6.3 ID Mapping of Data Items

This protocol is based on the principle that the client and the server can have their own ID's for data items in their databases. These ID's MAY or MAY NOT match with each other. Because the ID's can be different, the server MUST maintain the ID

mapping table for items. That is, the server knows which client ID (LUID) and which server ID (GUID) points to the same data item.

Figure 5 shows an example of an ID mapping table after synchronization. In this example the mapping table in the server is depicted as a separate from the actual database.

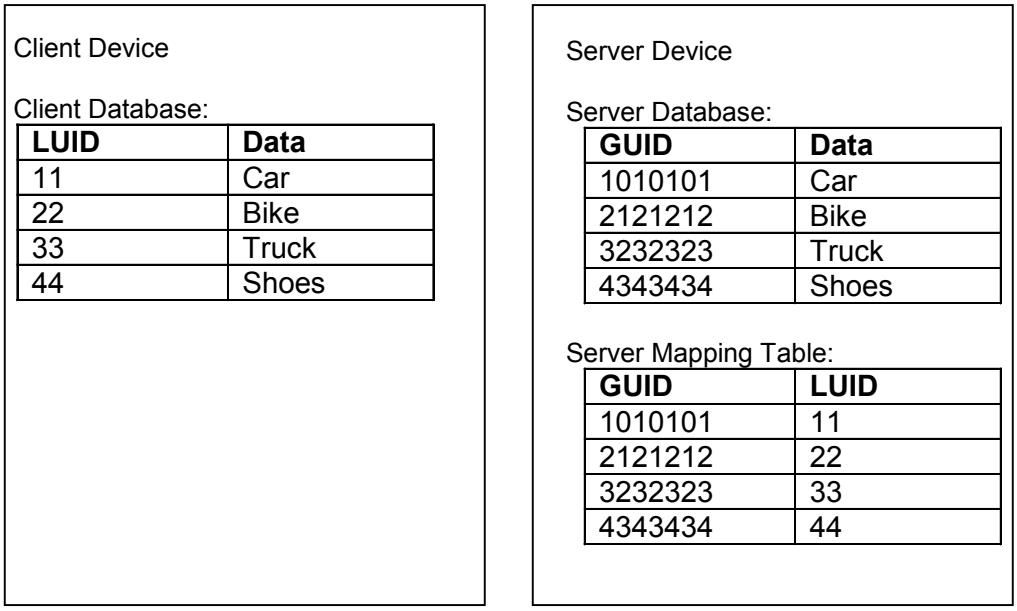

Figure 5 Example: ID Mapping of Data Items

The LUID's are always assigned by the client device. This means that even if the server adds an item to the client device, the client assigns a LUID for this item. In this case, the client returns the LUID of the new item to the server. The Map operation is used for this. After the Map operation is sent by the client, the server is able to update its mapping table with the client LUID.

When a server is adding a new item to a client, it MUST NOT send its actual GUID if the size of the actual GUID is exceeding the maximum size of the temporary GUID defined by the client. If size of the actual GUID's exceeds the maximum size, the server MUST use a smaller temporary GUID when adding an item to the client. The maximum size of the temporary GUID is defined in the device information document of the client.

If the server has modified an existing item (i.e., an item which is on both the devices), the server MUST identify the item by using the client LUID for this item, when the modification (e.g., replace or deletion) is synchronized with the client. In the case of the client modifications, items are also identified with LUID's, when the modifications are sent to the server. The server can map a LUID to its own GUID by utilizing the mapping table.

6.3.1 Caching of Map Operations

After an OMA DS server has requested one or more additions to be done by the OMA DS client, and the client has completed these additions to its database and allocated LUID's for them, the client has a possibility to cache the Map operations associated with these LUID's. The client MAY cache the Map operations, if the server has explicitly indicated that it does not require a response to its sync message. However, the client is always allowed to send the Map operations back to the server immediately after adding the items to the client database. This is the case even if the server has indicated that it does not require a response.

If the map items are cached, the Map operations are sent back to the server at the beginning of a subsequent synchronization session (in Pkg #3 from the client to the server). That is, the server MUST receive the Map operations before it is able to process any client updates related to the items with which the Map operations are associated.

If the OMA DS server has the control of a transport protocol (e.g., acting as a OBEX client), it MUST always request a response to the Sync command, which it has sent to the client. Thus, the server MUST NOT disconnect before it has got a response to the Sync command from the client.

6.4 Conflict Resolution

Conflicts happen because of modifications on the same items on the server and the client databases. (For example the same calendar item has been manually updated on the both sides.) In general a sync engine on the server resolves them. This protocol with the SyncML Representation protocol provides the functionality to notify the client about the resolved conflicts.

Although the server is in general assumed to include the sync engine functionality, the possibility that the client would also provide some sync engine functionality is not excluded. In this case, the client MAY also resolve conflicts. Then, the server only returns back to the client a notification that a conflict or conflicts have happened and the client can resolve the conflicts.

There are multiple policies to resolve the conflicts and the SyncML Representation protocol provides the status codes (See Chapter 10 in REPPRO) for some common policies. Thus, if the sync engine of the server resolves a conflict, it can send information about the conflict and how the conflict was resolved. This notification happens by using the Status elements. The example below depicts a case that the server sends a status to the client.

```
<Status>
   <CmdID>1</CmdID>
   <MsgRef>1</MsgRef>
   <CmdRef>2</CmdRef>
   <Cmd>Replace</Cmd>
   <SourceRef>1212</SourceRef>
   <Data>208</Data>        <!-- Conflict, originator wins -->
</Status>
```

The administration, and how the conflict resolution policy is configured, is out of the scope of this protocol and the SyncML Representation protocol.

6.5 Security

This protocol requires the support for the basic authentication and the MD5 digest access authentication on the server layer (i.e., in SyncHdr). Both the sync client and the server MAY initiate a challenge for authentication and the device receiving the authentication challenge MUST send the authorization credentials back.

The authentication procedure used by this protocol is defined in Chapter 7.

6.6 Addressing

6.6.1 Device and Service Addressing

The device or service addressing within the SyncML SyncHdr element is done by using the URI scheme defined in the SyncML representation specification. Devices connected to the Internet constantly, MAY refer to the URI-based addressing. E.g., the source would be:

```
<Source>
   <LocURI>http://www.syncml.org/sync-server</LocURI>
</Source>
```

Devices, which are, for example, connected temporarily, MAY prefer to identify themselves with their own identification mechanism. E.g., the Source element of a mobile phone device could be:

```
<Source>
   <LocURI>IMEI:493005100592800</LocURI>
</Source>
```

The addressing scheme on the transport level (e.g. HTTP) is independent from the addressing scheme used at the SyncML layer and the two schemes do not need to match.

6.6.2 Usage of RespURI and Re-direction Status Codes

Devices MUST support receiving the RespURI element as specified in the SyncML Representation Protocol specification, but the support of the re-direction status codes (3XX) is OPTIONAL.

6.6.3 Database Addressing

The database addressing within the SyncML operations is done by using the URI scheme defined in the SyncML Representation protocol. Absolute or relative URI's can be used for the server and client databases. E.g., the source elements for a server database in these two cases can look like:

```
<Sync>...
   <Target>
      <LocURI>./calendar/james_bond</LocURI>
   </Target>
   ...
</Sync>
<Sync>
   <Target>
      <LocURI>http://www.syncml.org/sync-server/calendar/james_bond</LocURI>
   </Target>
   ...
</Sync>
```

6.6.4 Addressing of Data Items

The addressing of data items within the SyncML Item elements is done by using the URI-based scheme defined in the SyncML representation specification. Relative URI's can be used. E.g., the source element for one item can look like:

```
<Item>
   ...
   <Source>
      <LocURI>101</LocURI>
   </Source>
   ...
</Item>
```

6.7 Exchange of Device Capabilities

This protocol provides the functionality to exchange the device capabilities during the initialization (See Chapter 8). The sync client or the sync server can request the exchange.

The sync client MUST send its device information to the server when the first synchronization is done with a server or when the static device information has been updated in the client. The client MUST also be able to transmit its device information if the server asks for it. The client SHOULD also support the functionality of receiving and processing the server device information and if it does MUST make use of it.

The sync server MUST be able to send its device information if requested by the client. The server MUST support the functionality of receiving and processing the client device information and MUST make use of it when sent by the client or requested by the server itself.

Implementation consideration. The exchange of the device information can require that quite a large amount of data be transferred over the air. Thus, the devices SHOULD avoid requesting the exchange every time when a sync is initialized. In addition, the devices SHOULD consider whether they need to send all device specific data if it is clear that another device cannot utilize it. E.g., if the client indicates that it does not support the vCard3.0 content format, the server SHOULD NOT send the supported properties of vCard3.0 if it supports it.

6.8 Device Memory Management

This protocol with the Meta Information DTD provides for the possibility to specify the dynamic memory capabilities for databases of a device or for persistent storage on a device. The capabilities specify how much memory there is left for usage.

The dynamic capabilities can be specified every time a synchronization is done. The static memory capabilities are exchanged when the sync initialization is done (See Chapter 6.7 and Chapter 8).

Although the sending of persistent memory capabilities is optional for both the sync clients and servers, the sync clients SHOULD send them and the sync servers MAY send them. If a sync client does send them then the sync server MUST respect them.

Below is an example of how the dynamic memory capabilities of a calendar database on a device are represented, when the Sync command is sent.

```
<Sync>
   <CmdID>1</CmdID>
   <Target><LocURI>./calendar/james_bond</LocURI></Target>
   <Source><LocURI>./dev-calendar</LocURI></Source>
   <Meta>
      <Mem xmlns='syncml:metinf'>
         <FreeMem>8100</FreeMem>
         <!--Free memory (bytes) in Calendar database on a device -->
         <FreeId>81</FreeId>
         <!--Number of free records in Calendar database-->
      </Mem>
   </Meta>
   <Replace>
      ...
   </Replace>
   ...
</Sync>
```

The database-specific memory elements in the Meta element of the Sync command MUST be associated with the source database specified in the Source element of the Sync command and MUST represent the amount of bytes that can be accepted in terms of the size taken up within the SyncML message.

6.9 Multiple Messages in Package

This protocol provides the functionality to transfer one SyncML package in multiple SyncML messages. This might be necessary if one SyncML package is too large to be transferred in one SyncML message. This limitation might be caused e.g., by the transport protocol or by the limitations of a small footprint device.

If a SyncML package is transferred in multiple SyncML messages, the last message in the package MUST include the Final element (See REPPRO). Other messages belonging to the package MUST NOT include the Final element. The Final element can only be included when all necessary commands belonging to a specific package have been sent. The final element MUST NOT be included if the other end has not closed the preceding package. E.g., if the server is still sending the package #4 to the client, the client MUST NOT close the package #5 prior to receiving the last message belonging to the package #4. The exclusion of the Final element is not to be used to indicate that a logical phase is not completed if an error occurs.

If a device receives a message in which the Final flag is missing and a Sync element for a database is included, the device MUST be able to handle the case that in the next message, there is another Sync element for the same database.

The device, which receives the SyncML package containing multiple messages, MUST be able to ask more messages. This happens by sending an Alert command with a specific alert code, 222 back to the originator of the package, or if there are other SyncML commands to be sent as a response, the Alert command with the 222 alert code MAY be omitted. After receiving the message containing the Final element, the Alert command MUST NOT be used anymore.

More messages are not desirable if errors, which prevent the continuation of synchronization, have occurred.

The receiver of a package MAY start to send its next package at the same time when asking more messages from the originator if this makes sense. Thus, in Chapters 8 - 12, it is specified which commands or elements are allowed to be sent before receiving the final message belonging to a package.

Below, there is depicted an example that the sync client is sending Package #3 in multiple messages (2 messages) and the server also sends Package #4 in multiple messages (2 messages).

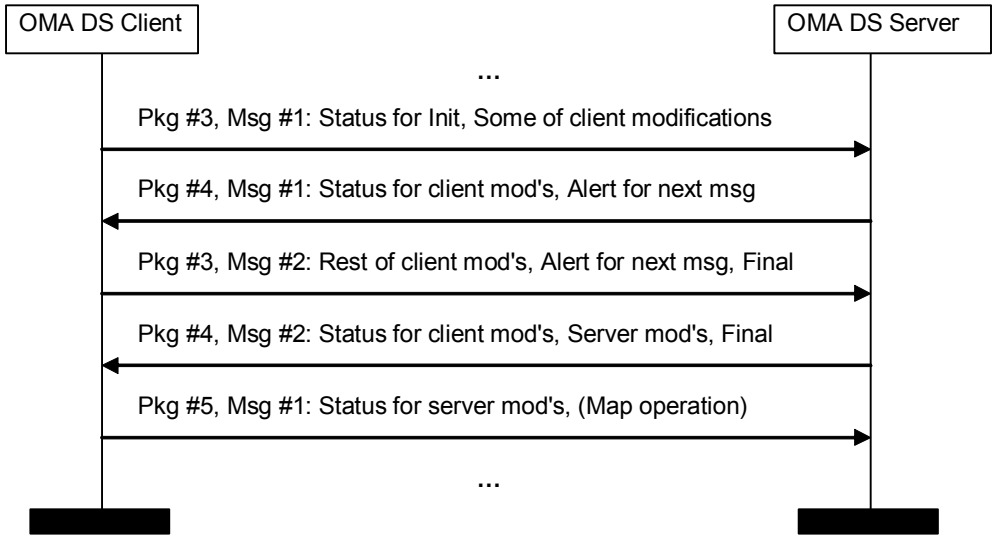

Figure 6 Example of Sending Multiple Messages in a Package

6.10 Large Object Handling

While synchronizing, object reception can be limited by two factors: the maximum *message* size the target device can receive (declared in <MaxMsgSize> tag), and the maximum *object* size the target device can receive (declared in <MaxObjSize> tag).

This feature provides a means to synchronize an object whose size exceeds that which can be transmitted within one message (e.g. the maximum message size – declared in <MaxMsgSize> element – that the target device can receive). This is achieved by splitting the object into chunks that will each fit within one message and by sending them contiguously. The first chunk of data is sent with the overall size of the object and a <MoreData/> signaling that more chunks will be sent. Every subsequent chunk is sent with a <MoreData/> tag, except from the last one: the final chunk is sent with no <MoreData/> tag. The target device, having received the final chunk, has to re-construct the object and consequently acts as it had received it in one piece (e.g. apply 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.

If an interruption occurs after either the client or server receives the first chunk of a large object, on resumption the overall remaining size of the object MUST be specified in the first chunk of data sent

NOTE 1: This mechanism does not allow sending multiple large objects in the same time. 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 Sync commands (i.e. Add, Replace, Delete, Copy, Atomic or Sequence) or new Items MUST NOT be placed between chunks of a data object.

NOTE 2: The overall remaining size sent in the first chunk of resumption is intended to allow resuming a session without the need to re-send the entire object.

6.10.1 Conformance statements:

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

Supporting Sending Large Objects is optional for both clients and servers.

A client supporting receiving Large Object MUST declare the <SupportLargeObjs/> tag in its DevInf.

Supporting receiving or sending Large Objects implies conformance constraints for several tags.

TAGS:

- **SupportLargObjs** [DEVINF]
- **MaxObjSize** [META]
- **MaxMsgSize** [META]
- **Size** [META]
- **MoreData** [REPPRO] – [DSREPU]

STATUS CODES AND ALERTS:

- **Status 213** Chunked item accepted and buffered [REPPRO]
- **Alert 222** NEXT MESSAGE [DSPROTO] – [DSREPU]
- **Alert 223** End of Data for chunked object not received [SYNCPRO] – [DSREPU]
- **Status 424** Size Mismatch [REPPRO]
- **Status 416** Request entity too large [REPPRO]
- **Status 411** Size REQUIRED. The requested command MUST be accompanied by byte size or length information in the `Meta` element type [REPPRO]

If a device supports receiving Large Objects it MUST declare the maximum size of object (<MaxObjSize> tag) it is capable of receiving as Meta information within the Alert or Sync command, as specified in [META].

The device MUST also declare and fill the <MaxMsgSize> tag. This tag, also declared as Meta Information, specifies the maximum byte size of any response message to a given request. Knowledge of both <MaxObjSize> and <MaxMsgSize> allows to compute appropriate data chunk size.

6.10.2 Large Object exchange sequence:

This section illustrates and details the process of Large Object Handling. The following figure depicts a normal flow when handling Large Objects between 2 entities: the "Sending Large Object Device" and the "Receiving Large Object Device". Note that these 2 entities can represent a Client or a Server: a Client can send Large Objects to a Server, but a Server can send also Large Objects to a Client.

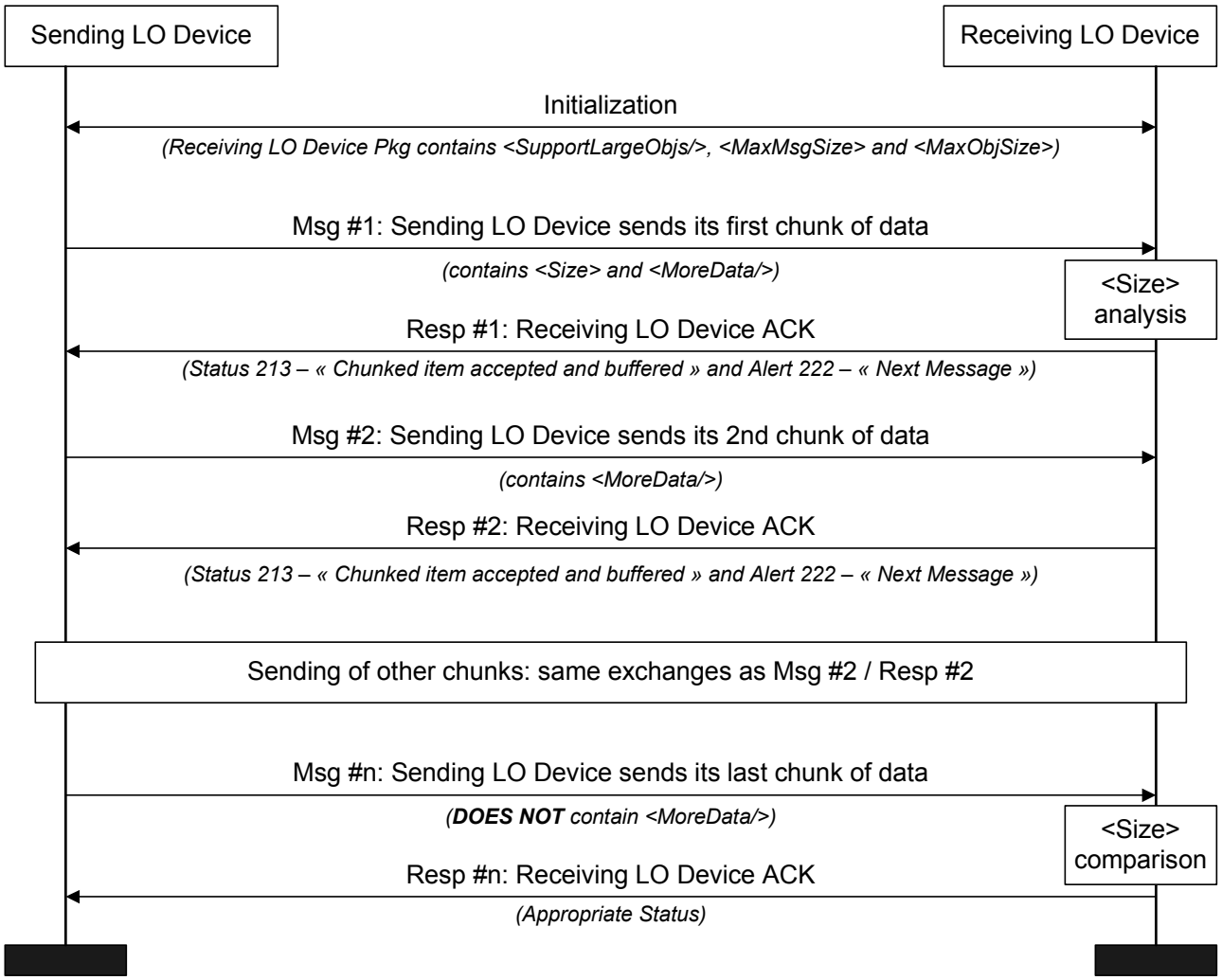

Figure 7 Example of Sending a Large Object (normal case)

NOTE 1: In the previous diagram, LO means "Large Object".

NOTE 2: Please refer to the section 6.9 for the use of the Alert 222.

Exchange of a Large Object can be summarized with the following sequence:

1. During initialization:

1.1. On the sending device side

1.1.1. Sending device SHOULD use knowledge of the recipient's **<MaxMsgSize>** to determine at what size segmentation occurs.

1.2. On the receiving device side

1.2.1. Receiving device MUST have declared the **<SupportLargeObjs/>** tag in its DevInf. It MUST also specify the value of its **<MaxMsgSize>** and its **<MaxObjSize>.**

2. When the first chunk of data is transmitted:

2.1. On the sending device side (Msg #1)

2.1.1. The sender MUST declare in the command element (e.g. add, replace) the overall size of the data element content that is going to be sent, using the **<Size>** sub-element of a Meta element.

NOTE: The <Size> element MUST only be specified for the first chunk of data. If an interruption occurs after the first chunk of a large object is received by either the client or server, on resumption the Meta tag, <Size>, MUST once again be specified in the first chunk being sent as part of the resumed session to indicate the size remaining.

2.1.2. A **<MoreData/>** empty element MUST be added after the <Data> element.

2.2. On the receiving device side (Resp #1)

2.2.1. On receipt of a data chunk with the **<MoreData/>** element, the recipient MUST respond with a **"Status 213 – Chunked item accepted and buffered"** and ask for the next message using the **Alert 222** mechanism as defined in section 6.9

Error case behavior:

1- If the Size exceeds the **<MaxObjSize>** of the recipient, the recipient MUST respond with a **"Status 416 - Requested size too big"** (the request failed because the specified byte size in the request was too big). The recipient MUST NOT commit the command.

2- If the recipient gets the first chunk with a **<MoreData/>** element, but no **<Size>** element, or non filled **<Size>** element, it MUST respond with a **"Status 411 - Size required"**. The recipient MUST NOT commit the command. The sender MAY attempt to retransmit the entire data object.

3. When extra chunks of data are transmitted:

3.1. On the sending device side (Msg #2)

3.1.1. Meta and Item information SHOULD be repeated on each subsequent message containing chunks of the same data object.

3.1.2. A **<MoreData/>** empty element MUST be added after the <Data> element.

3.2. On the receiving device side (Resp #2)

3.2.1. On receipt of a data chunk with the **<MoreData/>** element, the recipient MUST respond with a **"Status 213 – Chunked item accepted and buffered"** and ask for the next message using the **Alert 222** mechanism as defined in section 6.9

Error Case Behavior:

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 223 – End of Data for chunked object not received"**. The **Alert** SHOULD contain the complete 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 new and original commands. The sender MAY attempt to retransmit the entire original data object.

4. When the last chunk of data is transmitted:

4.1. On the sending device side (Msg #n)

4.1.1. The last chunk of data MUST NOT be followed with **<MoreData/>** element.

4.2. On the receiving device side (Resp #n)

4.2.1. On receipt of the last chunk of the data object, the recipient reconstructs the data object from its constituent chunks. It MUST validate that the size of re-constituted object matches the object **<Size>** supplied in the Meta information by the sender, then apply the requested command. The appropriate status MUST then be sent to the originator.

Error case behavior:

If the sizes do not match then a **"Status 424 – Size mismatch"** MUST be sent and the recipient MUST NOT commit the command. The sender MAY attempt to retransmit the entire data object.

6.10.3 Large Object exchange sequence example:

In this example the client sends a large object (for addition) to the server. The server has declared supporting Large Objects handling in its DevInf.

Client initializes a sync session

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1126272244708</SessionID>
      <MsgID>1</MsgID>
      <Target>
         <LocURI>http://Syncserver.com/sync</LocURI>
      </Target>
      <Source>
         <LocURI>IMEI_number</LocURI>
      </Source>
      <Cred>
         <Meta>
            <Format>b64</Format>
            <Type>syncml:auth-basic</Type>
         </Meta>
         <Data>dGVzdDp0ZXN0cHc=</Data>
      </Cred>
      <Meta>
         <MaxMsgSize>3000</MaxMsgSize>
      </Meta>
   </SyncHdr>
   <SyncBody>
      <Alert>
         <CmdID>1</CmdID>
         <Data>201</Data>
         <Item>
            <Target>
               <LocURI>./files</LocURI>
            </Target>
            <Source>
               <LocURI>file:///Files</LocURI>
            </Source>
            <Meta>
               <Anchor xmlns="syncml:metinf">
                  <Next>1126272244891</Next>
               </Anchor>
               <MaxObjSize>10000000</MaxObjSize>
            </Meta>
         </Item>
      </Alert>
      <Final/>
   </SyncBody>
```

```
</SyncML>
```

Server Response

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1126272244708</SessionID>
      <MsgID>1</MsgID>
      <Target>
         <LocURI>IMEI_number</LocURI>
      </Target>
      <Source>
         <LocURI> http://Syncserver.com/sync </LocURI>
      </Source>
      <Meta>
         <MaxMsgSize>30000</MaxMsgSize>
      </Meta>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>1</MsgRef>
         <CmdRef>0</CmdRef>
         <Cmd>SyncHdr</Cmd>
         <TargetRef> http://Syncserver.com/sync </TargetRef>
         <SourceRef>IMEI_number</SourceRef>
         <Data>212</Data>
      </Status>
      <Status>
         <CmdID>2</CmdID>
         <MsgRef>1</MsgRef>
         <CmdRef>1</CmdRef>
         <Cmd>Alert</Cmd>
         <TargetRef>./files</TargetRef>
         <SourceRef>file:///Files</SourceRef>
         <Data>200</Data>
         <Item>
            <Data>
               <Anchor xmlns="syncml:metinf">
                  <Next>1126272244891</Next>
               </Anchor>
            </Data>
         </Item>
      </Status>
      <Results>
         <!-- … -->
         <Item>
            <!-- … -->
            <Data>
               <DevInf xmlns='syncml:devinf'>
                  <!-- … -->
                  <SupportLargeObjs/>
                  <!-- … -->
               </DevInf>
            </Data>
         </Item>
      </Results>
      <Alert>
         <CmdID>3</CmdID>
         <Data>201</Data>
         <Item>
```

```
            <Target>
               <LocURI>file:///Files</LocURI>
            </Target>
            <Source>
               <LocURI>./files</LocURI>
            </Source>
            <Meta>
               <Anchor xmlns="syncml:metinf">
                  <Last>1126271088771</Last>
                  <Next>1126272246596</Next>
               </Anchor>
               <MaxObjSize>10000000</MaxObjSize>
            </Meta>
         </Item>
      </Alert>
      <Final/>
   </SyncBody>
</SyncML>
```

The client begins to send a large object

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1126272244708</SessionID>
      <MsgID>2</MsgID>
      <Target>
         <LocURI>http://Syncserver.com/sync?sid=W0JAMmYzNTZm</LocURI>
      </Target>
      <Source>
         <LocURI>IMEI_number</LocURI>
      </Source>
      <Meta>
         <MaxMsgSize>3000</MaxMsgSize>
      </Meta>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>1</MsgRef>
         <CmdRef>0</CmdRef>
         <Cmd>SyncHdr</Cmd>
         <TargetRef>IMEI_number</TargetRef>
         <SourceRef>http://Syncserver.com/sync?sid=W0JAMmYzNTZm</SourceRef>
         <Data>200</Data>
      </Status>
      <Status>
         <CmdID>2</CmdID>
         <MsgRef>1</MsgRef>
         <CmdRef>3</CmdRef>
         <Cmd>Alert</Cmd>
         <TargetRef>file:///Files</TargetRef>
         <SourceRef>./files</SourceRef>
         <Data>200</Data>
         <Item>
            <Data>
               <Anchor xmlns="syncml:metinf">
                  <Next>1126272246596</Next>
               </Anchor>
            </Data>
         </Item>
      </Status>
```

```
      <Sync>
         <CmdID>3</CmdID>
         <Target>
            <LocURI>./files</LocURI>
         </Target>
         <Source>
            <LocURI>file:///Files</LocURI>
         </Source>
         <Add>
            <CmdID>4</CmdID>
            <Meta>
               <Type>application/vnd.omads-file+xml</Type>
               <Size>2304</Size>
               <Version>20050909T094007Z</Version>
            </Meta>
            <Item>
               <Source>
                  <LocURI>p10.jpg</LocURI>
               </Source>
               <Data>
                  <!-- -->
                  <!-- Big Block Of Data Takes Place Here -->
                  <!-- -->
               </Data>
               <MoreData/>
            </Item>
            </Add>
         </Sync>
      </SyncBody>
</SyncML>
```

Server response : Data chunk is accepted

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1126272244708</SessionID>
      <MsgID>2</MsgID>
      <Target>
         <LocURI>IMEI_number</LocURI>
      </Target>
      <Source>
         <LocURI>http://Syncserver.com/sync?sid=W0JAMmYzNTZm</LocURI>
      </Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>2</MsgRef>
         <CmdRef>0</CmdRef>
         <Cmd>SyncHdr</Cmd>
         <TargetRef>http://Syncserver.com/sync</TargetRef>
         <SourceRef>IMEI_number</SourceRef>
         <Data>200</Data>
      </Status>
      <Status>
         <CmdID>2</CmdID>
         <MsgRef>2</MsgRef>
         <CmdRef>3</CmdRef>
         <Cmd>Sync</Cmd>
         <TargetRef>./files</TargetRef>
         <SourceRef>file:///Files</SourceRef>
```

```
         <Data>200</Data>
      </Status>
      <Status>
         <CmdID>3</CmdID>
         <MsgRef>2</MsgRef>
         <CmdRef>4</CmdRef>
         <Cmd>Add</Cmd>
         <SourceRef>p10.jpg</SourceRef>
         <Data>213</Data>
      </Status>
      <Alert>
         <CmdID>4</CmdID>
         <Data>222</Data>
         <Item>
            <!-- … -->
         </Item>
      </Alert>
   </SyncBody>
</SyncML>
```

…

Client sends the last chunk of the large object

```
<SyncML xmlns="SYNCML:SYNCML1.2">
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1126272244708</SessionID>
      <MsgID>3</MsgID>
      <Target>
         <LocURI>http://Syncserver.com/sync?sid=W0JAMmYzNTZm</LocURI>
      </Target>
      <Source>
         <LocURI>IMEI_number</LocURI>
      </Source>
      <Meta>
         <MaxMsgSize>3000</MaxMsgSize>
      </Meta>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>2</MsgRef>
         <CmdRef>0</CmdRef>
         <Cmd>SyncHdr</Cmd>
         <TargetRef>IMEI_number</TargetRef>
         <SourceRef>http://Syncserver.com/sync?sid=W0JAMmYzNTZm</SourceRef>
         <Data>200</Data>
      </Status>
      <Alert>
         <CmdID>2</CmdID>
         <Data>222</Data>
         <Item>
            <Data>Next Message Please</Data>
         </Item>
      </Alert>
      <Sync>
         <CmdID>3</CmdID>
         <Target>
            <LocURI>./files</LocURI>
         </Target>
         <Source>
```

```
            <LocURI>file:///Files</LocURI>
          </Source>
          <Add>
            <CmdID>4</CmdID>
            <Meta>
              <Type>application/vnd.omads-file+xml</Type>
              <Version>20050909T094007Z</Version>
            </Meta>
            <Item>
              <Source>
                <LocURI>p10.jpg</LocURI>
              </Source>
              <Data>
                <!-- -->
                <!-- Large Object Ends Here -->
                <!-- -->
              </Data>
            </Item>
          </Add>
        </Sync>
        <Final/>
    </SyncBody>
</SyncML>
```

…..

6.11 Hierarchical synchronization

Hierarchical synchronization consists in synchronizing a hierarchical data structure on a server and its equivalent on the client. A hierarchical data structure can be compared to a tree structure that is composed of:

- Branches that are links between a node and its children
- Nodes that contain the information : There are three kind of nodes :
  - the root node which has no parent
  - internal node which has a unique parent
  - terminal node (or leaf) a node with no children

One of the most known tree structure is the filesystem where folders act as nodes and files as leaves

Regarding OMA-DS protocol, hierarchical synchronization mechanism is based on the use of <TargetParent> and <SourceParent> element.

The client uses <SourceParent> to specify the LUID for the parent of the client's side item.

Depending on the existence of the parent item on the client, the server will use TargetParent or SourceParent :

- If the parent exists on the client , the server must use the TargetParent tag that will contain the client's LUID for the parent of the server side's item
- If the parent item doesn't exist on the client, the server must use the SourceParent tag that will contain a temporary GUID for the parent of the server side's item. This case occurs when the server send items that the client has not mapped yet (eg moving a file into a newly created folder)

As an example we propose in the following a possible filesystem synchronization scenario

We suppose the client and server are synchronized. Items mapping table is

Client's LUID	Server's GUID	Object name
990	ABCD990	Urgent
995	ABCD995	Work
1000	ABCD1000	Image1.jpg
1001	ABCD1001	Pictures
1002	ABCD1002	Friends

After the last synchronization the following modifications were made on the client:

- The folder "NewFolder" has been created at the root of the filesystem.
- A file named "NewDocument.doc" has been created in the "NewFolder" folder
- The file "Image1.jpg" has been moved to the "Pictures" folder

Regarding the server modifications made are:

- A new folder "ToBeDone" has been created in the folder "Work"
- A file "ToDoList.doc" has been created in the folder "ToBeDone"
- Subfolder "Urgent" has been moved to "ToBeDone"

For those new resources the partial mapping table can be expressed as:

Client's LUID	Server's GUID	Object name
	ABCD997	ToBeDone
	ABCD998	ToDoList.doc
1003		NewFolder
1004		NewDocument.doc

A subsequent synchronization will produce the following package snippets :

Package 3 : Client modifications

```
...
<Sync>
   <CmdID>7</CmdID>
   <Target>
      <LocURI>./FileSystem</LocURI></Target>
   <Source>
      <LocURI>Files</LocURI></Source>
   <Add>
      <CmdID>8</CmdID>
      <Meta>
         <Type xmlns='syncml:metinf'>application/vnd.omads-folder+xml</Type>
```

```
      </Meta>
      <Item>
         <Source><LocURI>1003</LocURI></Source>
         <SourceParent><LocURI>/</LocURI></SourceParent>
         <Data> Data containing DataObjFolder should be placed here </Data>
      </Item>
   </Add>
   <Add>
      <CmdID>9</CmdID>
      <Meta>
         <Type xmlns='syncml:metinf'>application/vnd.omads-file+xml</Type>
      </Meta>
      <Item>
         <Source><LocURI>1004</LocURI></Source>
         <SourceParent><LocURI>1003</LocURI></SourceParent>
         <Data>Data containing DataObjFile should be placed here</Data>
      </Item>
   </Add>
   <Move>
      <CmdID>10</CmdID>
      <Meta>
         <Type xmlns='syncml:metinf'>application/vnd.omads-file+xml</Type>
         <Item>
            <Source><LocURI>1000</LocURI></Source>
            <SourceParent><LocURI>1001</LocURI></SourceParent>
         </Item>
      </Meta>
   </Move>
</Sync>
...
```

The client modification package contains:

- An <Add> command with SourceParent containing "/" and <Source> "1003" (creation of "NewFolder" at the root)
- An <Add> command with SourceParent containing "1003" and <Source> "1004" (creation of "NewDocument.doc" int the "NewFolder" folder)
- A <Move> command with SourceParent containing "1001" and <Source> "1000" (Moving of "image1.jpg" in "Pictures")

Package 4: server modifications

```
...
<Sync>
   <CmdID>17</CmdID>
   <Target><LocURI>Files</LocURI></Target>
   <Source><LocURI>./FileSystem </LocURI></Source>
   <Add>
      <CmdID>18</CmdID>
      <Meta>
         <Type xmlns='syncml:metinf'>application/vnd.omads-folder+xml</Type>
      </Meta>
      <Item>
         <Source><LocURI>ABCD997</LocURI></Source>
         <TargetParent><LocURI>995</LocURI></TargetParent>
         <Data> Data containing DataObjFolder should be placed here </Data>
      </Item>
   </Add>
   <Add>
```

```
      <CmdID>19</CmdID>
      <Meta>
         <Type xmlns='syncml:metinf'>application/vnd.omads-file+xml</Type>
         <Item>
            <Source><LocURI>ABCD998</LocURI></Source>
            <SourceParent><LocURI>ABCD997</LocURI></SourceParent>
            <Data> Data containing DataObjFile should be placed here </Data>
         </Item>
      </Meta>
   </Add>
   <Move>
      <CmdID>20</CmdID>
      <Meta>
         <Type xmlns='syncml:metinf'>application/vnd.omads-folder+xml</Type>
      <Item>
         <Target><LocURI>990</LocURI></Target>
         <SourceParent><LocURI>ABCD997</LocURI></SourceParent>
      </Item>
   </Move>
</Sync>
...
```

The server modifications package contains:

- An <Add> command with <TargetParent> containing "995" (the client's LUID for the parent of the "ToBeDone" item)
- An <Add> command with <SourceParent> containing "ABCD997" (Since the client has not mapped "ToBeDone" yet we use <SourceParent> and the server side's parent GUID of "ToDoList.doc")
- A move command with <SourceParent> containing "ABCD997" and <Target> containing 990 (Moving "Urgent" to "ToBeDone")

6.12 Sync without Separate Initialization

Description:
Synchronization can be started without a separate initialization (See the initialization in Chapter 8). This means that the initialization is done simultaneously with sync. This can be done to decrease the number of SyncML messages to be sent over the air.

Conformance statement:
Clients MAY support the feature "Sync without separate initialization".
Servers MUST support the feature "Sync without separate initialization".

Behaviour description when client and server do implement the feature:
When the client initiates the synchronization session, it combines the Package #1 within the Package #3. The Alert command(s) (from the client) in Package #1 is sent within Package #3, in which the Sync command(s) are also placed. The server MUST combine Package #2 within Package #4. The Alert command(s) (from Server) in Package #2 is sent within Package #4, in which the Sync command(s) are also placed.

See the example in Examples.

6.12.1 Robustness and Security Considerations

If the client implementation decides to use sync without a separate initialization, the following considerations SHOULD be taken into account:

- The client sends its modifications to the server before the server gets the sync anchors from the client. Because of this, the client might need to send all data again if the server has a need to request a slow sync.

- Server sync anchor are not received before sending the client modifications. Thus, if the client needs to request a slow sync, earlier data, which was sent in Package #3 to the server, was unnecessarily sent and all data needs to be sent to server.
- The client sends its modifications to the server before there is any possibility for the server to send its credentials (if requested) to the client. I.e., the client cannot be sure whether it is communicating with the correct server.

6.12.2 Example of Sync without Separate Initialization

Here is shown an example, how the client starts sync without a separate sync initialization. Only two packets are shown here (combination of Packages #1 and #3 and the combination of Packages #2 and #4). Package #5 and #6 can follow as defined in the specification.

Combination of Package #1 and #3

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>4</SessionID>
      <MsgID>1</MsgID>
      <Target><LocURI>http://www.syncml.org/sync-server</LocURI></Target>
      <Source><LocURI>IMEI:493005100592800</LocURI></Source>
      <Cred> <!--The authentication is optional.-->
         <Meta><Type xmlns='syncml:metinf'>syncml:auth-basic</Type></Meta>
         <Data>QnJ1Y2UyOk9oQmVoYXZl</Data> <!--base64 formatting of "userid:password"-->
      </Cred>
   </SyncHdr>
   <SyncBody>
      <Alert>
         <CmdID>1</CmdID>
         <Data>200</Data> <!-- 200 = TWO_WAY_ALERT -->
         <Item>
            <Target><LocURI>./contacts/james_bond</LocURI></Target>
            <Source><LocURI>./dev-contacts</LocURI></Source>
            <Meta>
               <Anchor xmlns='syncml:metinf'>
                  <Last>234</Last>
                  <Next>276</Next>
               </Anchor>
            </Meta>
         </Item>
      </Alert>
      <Sync>
         <CmdID>2</CmdID>
         <Target><LocURI>./contacts/james_bond</LocURI></Target>
         <Source><LocURI>./dev-contacts</LocURI></Source>
         <Meta>
            <Mem xmlns='syncml:metinf'>
               <FreeMem>8100</FreeMem>
               <!--Free memory (bytes) in Calendar database on a device -->
               <FreeId>81</FreeId>
               <!--Number of free records in Calendar database-->
            </Mem>
         </Meta>
         <NumberOfChanges>1</NumberOfChanges>
         <Replace>
            <CmdID>3</CmdID>
            <Meta><Type xmlns='syncml:metinf'>text/x-vcard</Type></Meta>
            <Item>
               <Source><LocURI>1012</LocURI></Source>
```

```
            <Data><!--The vCard data would be placed here.--></Data>
          </Item>
        </Replace>
      </Sync>
      <Final/>
   </SyncBody>
</SyncML>
```

Combination of Package #2 and #4

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>4</SessionID>
      <MsgID>1</MsgID>
      <Target><LocURI>IMEI:493005100592800</LocURI></Target>
      <Source><LocURI>http://www.syncml.org/sync-server</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>1</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
         <TargetRef>http://www.syncml.org/sync-server</TargetRef>
         <SourceRef>IMEI:493005100592800</SourceRef>
         <Data>212</Data> <!--Statuscode for OK, authenticated for session-->
      </Status>
      <Status>
         <CmdID>2</CmdID>
         <MsgRef>1</MsgRef><CmdRef>1</CmdRef><Cmd>Alert</Cmd>
         <TargetRef>./contacts/james_bond</TargetRef>
         <SourceRef>./dev-contacts</SourceRef>
         <Data>200</Data> <!--Statuscode for OK-->
         <Item>
            <Data><Anchor xmlns='syncml:metinf'><Next>276</Next></Anchor></Data>
         </Item>
      </Status>
      <Status>
         <CmdID>3</CmdID>
         <MsgRef>1</MsgRef><CmdRef>2</CmdRef><Cmd>Sync</Cmd>
         <TargetRef>./contacts/james_bond</TargetRef>
         <SourceRef>./dev-contacts</SourceRef>
         <Data>200</Data> <!--Statuscode for Success-->
      </Status>
      <Status>
         <CmdID>4</CmdID>
         <MsgRef>1</MsgRef><CmdRef>3</CmdRef><Cmd>Replace</Cmd>
         <SourceRef>1012</SourceRef>
         <Data>200</Data> <!--Statuscode for Success-->
      </Status>
      <Alert>
         <CmdID>5</CmdID>
         <Data>200</Data> <!-- 200 = TWO_WAY_ALERT -->
         <Item>
            <Target><LocURI>./dev-contacts</LocURI></Target>
            <Source><LocURI>./contacts/james_bond</LocURI></Source>
            <Meta>
               <Anchor xmlns='syncml:metinf'>
         <Last>20040119T081812Z </Last>
         <Next>20040120T093223Z </Next>
               </Anchor>
            </Meta>
         </Item>
      </Alert>
      <Sync>
```

```
        <CmdID>6</CmdID>
        <Target><LocURI>./dev-contacts</LocURI></Target>
        <Source><LocURI>./contacts/james_bond</LocURI></Source>
        <NumberOfChanges>2</NumberOfChanges>
        <Replace>
           <CmdID>7</CmdID>
           <Meta><Type xmlns='syncml:metinf'>text/x-vcard</Type></Meta>
           <Item>
              <Target><LocURI>1023</LocURI></Target>
              <Data><!--The vCard data would be placed here.--></Data>
           </Item>
        </Replace>
        <Add>
           <CmdID>8</CmdID>
           <Meta><Type xmlns='syncml:metinf'>text/x-vcard</Type></Meta>
           <Item>
              <Source><LocURI>10536681</LocURI></Source>
              <Data><!--The vCard data would be placed here.--></Data>
           </Item>
        </Add>
     </Sync>
     <Final/>
  </SyncBody>
</SyncML>
```

6.13 Suspend and Resume of synchronization session

When a sync session is interrupted it is possible to resume the session from where it was interrupted.

Interruptions can occur in two different ways:

1. User initiated interruption/Pause (Can also be viewed as an intentional pause):

   This kind of interruption occurs when a user requests to pause the current session, thereby resulting into a negotiation phase between client and server to pause the session.

   In order to interrupt the sync session, the client MAY send a message containing 'Interrupt Sync Session' (Alert 224 SUSPEND) with no client side data items and MAY contain statuses to server's data items. This message can be sent before receiving the complete package from the server.

2. Loss of network coverage or phone malfunction (Can also be viewed as an unintentional pause):

   This kind of interruption can be due to loss of network coverage or phone malfunction or for other unknown reasons that lead to an immediate interruption of the sync session and thus not needing a special alert code for interruption.

The interruption MAY occur during any of the synchronization phases - initialization, or during synchronization, or during mapping. Regardless of which type of interruption occurs however, the mechanism to resume a session is the same (Alert 225 RESUME).

Implementor's note: When an intentional interruption occurs, which corresponds to the "Suspend" mechanism, the intention is for the client to acknowledge as many operations as possible which have occurred prior to the interruption so as to reduce the number of operations that will need repeating once the session is resumed.

Although it is the client's responsibility to request a session resumption it is the server that has the final say as to wether a session can truly be resumed. The server always has the prerogative to refuse a session resumption (Status 508 REFRESH REQUIRED).

The sections that follow more thoroughly explain how sessions can be suspended and resumed.

6.13.1 Interrupting a synchronization session

As mentioned above a sync session can be interrupted either intentionally or unintentionally.

The Suspend mechanism allows the client ***and only the client*** to alert the server that an interruption is going to occur. This interruption can occur at anytime during the sync session. When suspending a session intentionally in this manner the client SHOULD attempt to acknowledge the operations already sent prior to the interruption in order to limit the number of operations that may end up being re-sent during a future resumption.

The flow for an intentional suspend should therefore proceed as follows:

1. The client sends a message with an Alert 224.
2. The server answers to this message by acknowledging the Alert with a "Status 200".
3. The session is interrupted.

The following figure illustrates the "Suspend" mechanism:

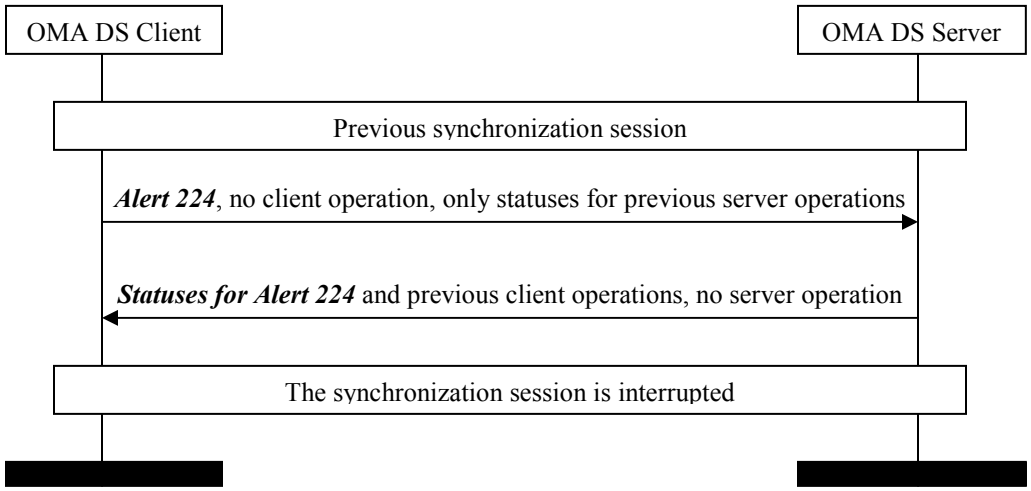

Figure 8: Suspend mechanism

Implementer's Note:

When requesting the suspend (Alert 224) consider the following:

- *It makes no sense for the message to contain any operations from the client for the server since a suspension is being requested so the server won't be able to act on them.*
- *An attempt to acknowledge in this message all the operations that it has already received from the server and dealt with does make sense since it will help reduce what may need to be resent during a future resumption.*
- *The client shouldn't perform other server operations after sending the suspension request.*
- *Invoking an intentional suspension during the last package (pkg #6) with the "Suspend" mechanism makes no sense since the sync is done. It makes more sense to just wait for the end of the sync session.*

When responding to the suspension request consider the following:

- *It makes no sense for the message to contain any more operations for the client since a suspension is being requested so the client won't be able to act on them.*
- *An attempt to acknowledge in this message all the operations that the server has already received from the client and dealt with does make sense since it will help reduce what may need to be resent during a future resumption.*
- *The server shouldn't perform other client operations after this moment.*

The following examples illustrate an interruption for both slow sync and normal sync with the help of SyncML messages.

i Client device initiate synchronization of Contacts, Calendar datastore. Due to maximum message size, client is able to send all contacts modifications, in this example 10 contacts items, but only 5 of 10 calendar modifications.

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
      <Target>
         <LocURI>http://syncserver.com/servlets/SyncML</LocURI>
      </Target>
      <Source>
         <LocURI>IMEI:004400061769830</LocURI>
      </Source>
      ...
   </SyncHdr>
   <SyncBody>
      <Status>
         <!-- Statuses for SyncHdr and so on --!>
         ...
      </Status>
      <Sync>
         <CmdID>4</CmdID>
         <Target>
            <LocURI>./Contact/Contacts</LocURI>
         </Target>
         <Source>
            <LocURI>./C\System\Data\Contacts.cdb</LocURI>
         </Source>
         ...
         <!--10/10 Contact items-->
      </Sync>
      <Sync>
         <CmdID>15</CmdID>
         <Target>
            <LocURI>./Calendar/Calendars</LocURI>
         </Target>
         <Source>
            <LocURI>./C\System\Data\Calendars.cdb</LocURI>
         </Source>
         ...
         <!-- 5/10 Calendar items-->
      </Sync>
   </SyncBody>
</SyncML>
```

ii Server responds with statuses of client's items that were sent in the previous example by the client.

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
      <Target>
         <LocURI>IMEI:004400061769830</LocURI>
      </Target>
      <Source>
         <LocURI>http://syncserver.com/servlets/SyncML</LocURI>
      </Source>
      ...
   </SyncHdr>
   <SyncBody>
      <Status>
         <!-- Statuses for SyncHdr, Sync, 10/10 contact items and 5/10 calendar items --!>
         ...
      </Status>
   </SyncBody>
</SyncML>
```

iii User decides to interrupt the sync session and so instead of sending the remaining calendar items, client alerts the server to interrupt the session This interruption is an intentional interruption where the user request for pausing the ongoing synchronization session. Hence, client MUST sent an Alert Code – 224, 'Interrupt Session '. The alert message from client to interrupt sync session MAY also contain statuses to server's data items.

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>3</MsgID>
      <Target>
         <LocURI>http://syncserver.com/servlets/SyncML</LocURI>
      </Target>
      <Source>
         <LocURI>IMEI:004400061769830</LocURI>
      </Source>
      ...
   </SyncHdr>
   <SyncBody>
      <Status>
         <!-- Statuses for SyncHdr and so on --!>
         ...
      </Status>
      <Alert>
         <CmdID>1</CmdID>
         <Data>224</Data> <!-- Alert code to 'Interrupt Session' -->
         <Item>
            <Target>
               <LocURI>http://syncserver.com/servlets/SyncML</LocURI>
            </Target>
            <Source>
               <LocURI>>http://IMEI:004400061769830</LocURI>
            </Source>
         </Item>
      </Alert>
      <Status>
         <CmdID>2</CmdID>
         <MsgRef>2</MsgRef>
         <CmdRef>0</CmdRef>
         <Cmd>SyncHdr</Cmd>
```

```
        <TargetRef>IMEI:004400061769830</TargetRef>
        <SourceRef>http://syncserver.com/servlets/SyncML</SourceRef>
        <Data>200</Data>
      </Status>
   </SyncBody>
</SyncML>
```

iv Server accepts client's request to interrupt the session by acknowledging to the client's alert request.

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>3</MsgID>
      <Target>
         <LocURI>IMEI:004400061769830</LocURI>
      </Target>
      <Source>
         <LocURI>http://syncserver.com/servlets/SyncML</LocURI>
      </Source>
      ...
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>3</MsgRef>
         <CmdRef>0</CmdRef>
         <Cmd>SyncHdr</Cmd>
         <TargetRef>http://syncserver.com/servlets/SyncML</TargetRef>
         <SourceRef>IMEI:004400061769830</SourceRef>
         <Data>200</Data>
      </Status>
      <Status>
         <CmdID>2</CmdID>
         <MsgRef>3</MsgRef>
         <CmdRef>1</CmdRef>
         <Cmd>Alert</Cmd>
         <TargetRef>http://syncserver.com/servlets/SyncML</TargetRef>
         <SourceRef>IMEI:004400061769830</SourceRef>
         <Data>200</Data>
      </Status>
   </SyncBody>
</SyncML>
```

At this point client and server can terminate the session by disconnecting the transport. Since this is an interruption of the session the 'Last Anchor' MUST NOT be updated. Also, client and server MUST record that the session was interrupted so that the next session will be a resume instead of a new session. In the case of an unintentional interruption, the "Suspend" mechanism is not used; nevertheless this still counts as an interruption of the session so the same would be expected. Of course, if for some reason the client wishes to force a slow-sync then it MAY do so by not sending an alert to resume.

6.13.2 Resuming synchronization session

The Resume mechanism provides the possibility to re-start an interrupted session without re-sending all the operations that have received Statuses during the previous sync session. All the operations of the previous sync session that have not been acknowledged with a Status must be re-sent during the resumption.

Whether an interruption occurs intentionally or unintentionally, a session can be resumed and the Resume mechanism is strictly the same for these two types of interruption. The resumption of a session is allowed for all Sync types and is identical. The client is always the originator of a resumption.

The following figure illustrates the "Resume" mechanism when the resumption is accepted by the two sides (normal case):

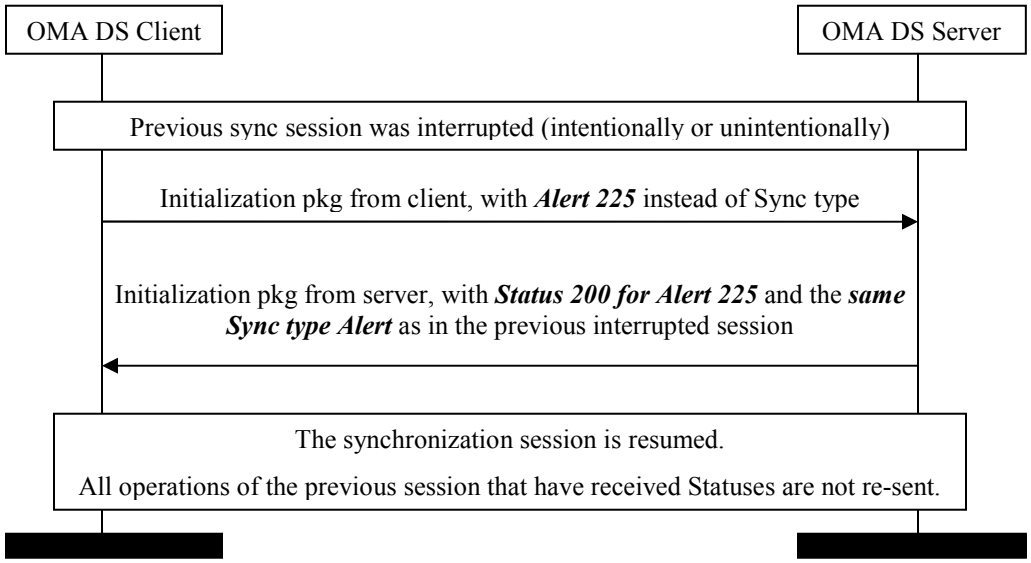

Figure 9: Resume mechanism

Resuming a session:

The "Resume" mechanism is composed of the following steps in the case where both client and server accept the resumption (normal case):

1. The client sends its initialization package for a new session with an Alert 225 instead of the Sync type. The ID used for this session MUST be different from the previous interrupted one. The Last Anchor value MUST NOT have been updated since the previous session and does not need to be sent. The Next Anchor MUST be updated and MUST be sent.
2. The server answers to this message by acknowledging the Alert 225 with a "Status 200" in its initialization package. This package MUST also contain an Alert with the ***same sync type code*** used in the previous interrupted session. As for the client, the Last Anchor value MUST NOT have been updated since the previous session and does not need to be sent. The Next Anchor MUST be updated and MUST be sent.
3. The session is resumed: the remaining operations of the previous session are sent (i.e. all the previous operations that have not been acknowledged with a Status). The operations that have received Statuses during the previous session MUST NOT be re-sent.

What must be sent during a resumption:

The operations that must be sent or re-sent during the resumption depend on the moment when the interruption has occurred, considering in addition that some modifications can occur at both sides during the interruption:

Case 1: *The interruption has occurred during the initialization phase*

(Pkg #1 or Pkg #2)

Additional requirements: instead of resuming, the client MUST launch a new sync session with the same sync type as the previous attempt, since no operation has been sent during the previous session. Therefore, all the operations MUST be sent again during the new sync session. All the modifications that have occurred at both sides during the interruption MUST also be taken into account during the new sync session.

Implementors Note: Since such a suspension occurs before anything has actually happened which could change the state of either the client or server since the last time they sync'd it should be treated as if if it never happened.

The client should simply try again. There is nothing really to resume. Neither Clients nor Servers should consider the need for a slow sync in such situations.

Case 2: *The interruption has occurred during the phase where the client was sending its modifications to the server (Pkg #3)*

During the resumption, the client MUST send the remainder of package #3, i.e. all the client operations that the server has not acknowledged with a Status during the first interrupted sync session. Additional requirements: the client MUST include inside the resumed session all the modifications that have occurred client side during the interruption. The server MUST send all its operations during the resumed session, including all the modifications that have occurred server side during the interruption.

Case 3: *The interruption has occurred during the phase where the server was sending its modifications to the client (Pkg #4)*

During the resumption, the server MUST send the remainder of the package #4, i.e. all the server operations that the client has not acknowledged with a Status during the first interrupted sync session.

Additional requirements:

--the server MUST include inside the resumed session all the modifications that have occurred server side during the interruption.

--Servers MUST maintain the temporary identifiers for the entire synchronization. This means when a session is interrupted during an Add operation from server to client, during resuming the session, server MUST use the same temporary identifiers as used in the interrupted session. Also, if an Add from server to the client has been successful and if the session is interrupted then client MUST send the 'Map' operation during the resume session. (See the reference to "Caching of Maps" in Case 4 below)

--The client MUST NOT include inside the resumed session the modifications that have occurred client side during the interruption (as Pkg #3 has already been successfully processed). The client is expected to send an empty pkg #3 with a final tag. * In this case, two situations are possible:

Conflict with the server: The client modifications that have occurred during the interruption lead to a conflict when the server sends to the client the rest of the pkg #4. For example, the user has modified an item's field that the server also modifies with a command during the resumption. In this situation,

-either the client decides not to let the server modifications win, and sends status code (409) (Conflict. The requested command failed because of an update conflict between the client and server versions of the data) in response to the server command for the item that was the source of the conflict, and continues to process the rest of the updates, then afterward, launches a new 2-way sync with the conflicting data item(s).

- OR the client decides to let the server modification supersedes its own modifications, and end up in the situation below (no conflict anymore) and MUST act as described below :

No conflict with the server: The client modifications that have occurred during the interruption do not lead to a conflict when the server sends to the client the rest of the pkg #4. In this situation, the client MUST launch a new Two Way Sync session just after the resumption, in order to synchronize these client modifications, and to ensure that the client and the server data stores will be equivalent.

***Implementor's note: The decision to ban sending client updates in the resumed session after server has started to send pkg #4 was made in 1.2 to avoid the potential ambiguities of some of the edge cases where both client and server have updated the same item.**

Case 4: *The interruption has occurred during the Mapping phase*

(Pkg #5 or Pkg #6)

Additional requirements:

- The client MUST use the Alert code 225 as specified in the "Resume" mechanism, in order to resume the session;
- The client MUST send the remainder of the Map operations inside the resumed session, i.e. all the Map operations that have not been acknowledged by the server with a Status during the first interrupted session, ***using the mechanism defined in the section "Caching of Maps Operations"*** (see [DSPRO-1.2], section 6.3.1: *"The Map operations are sent back to the server at the beginning of a subsequent synchronization session (in Pkg #3 from the client to the server)."*)
- Both client and server MAY include inside the resumed session all the modifications that have occurred at the client side or at the server side during the interruption. The Mapping regarding these new modifications MUST be done at the end of the resumed session. OR the client MAY allow the seesion to end fully and then initiate a new sync to send the new changes.

Refusing to Resume a session:

Additional requirements: the client and the server MUST always request a Slow Sync type after an interrupted session which is not resumed, in order to ensure that the two data stores will be consistent after the sync session.

Either the client or the server can refuse resuming the session, but the mechanism is not the same for the two sides:

- The client is allowed not to resume an interrupted session. In that case, the client initiates a new session with a Slow Sync type, by inserting the appropriate Alert code 201 instead of the Alert code 225. Additional requirement: the client MUST NOT request a sync type different from a Slow Sync in that case. When the Alert code 225 is not used, the server MUST understand that the session is not resumed, and that a new Slow Sync session is starting.
- The server is allowed to refuse to accept the client's request to resume an interrupted session. In that case, the server answers to the Alert 225 sent by the client with a Status 508 "Refresh required" in the server initialization package. Additional requirement: the server MUST use **either** the Alert code 201 (Slow Sync**) or 205 (Refresh Required)** in this server's package in order to force a Refresh in that case. After the reception of a Status code 508, the client MUST understand that the server refuses to resume the session, and that a new Slow Sync **or Refresh** session is starting.

The following table summarizes the different Status and Alert codes that the server can potentially use in its answer to a Resume request sent by the client, as well as the expected client behaviour:

Status code	Alert code	Meaning	Client behaviour
200	Same Sync type code as in the previous interrupted session	The resumption of the session is accepted by both client and server	The client MUST resume the interrupted session.
200	Sync type code different from the previous interrupted session one	**NOT POSSIBLE** ***The server is not OMA DS 1.2 compliant in that case***	The client MUST end the resumed session in failure with a relevant error status code
508	Slow Sync type code 201	The server refuses to resume the session and forces a Slow Sync.	The client MUST perform a Slow Sync
508	Sync type code 205	The server refuses to resume the session and indicates the new Sync type it wants to use. **In an enterprise**	The client MAY accept the override or end the resumed session in failure with a relevant error status code

		environment, there are times when the server would prefer a "Refresh from Server" type of Sync (Alert 205) rather than a full Slow Sync (Alert 201). The use case for this is when the client device type is known to corrupt data (eg. May use vcal only and may introduce oddities in representation of exceptions to recurring events which MUST NOT be propagated to the server datastore.)	

Table 2: Status and Alert codes used by the server when the client tries to resume a session

The following examples illustrate resumption for both slow sync and normal sync with the help of SyncML messages.

Resuming of the sync session can be indicated by using an Alert code for 'Resume session' during the initialization phase i.e. package 1 and package 2.

Client initiated slow-sync:

```
<?xml version="1.0" encoding="UTF-8"?>
<SyncML xmlns='SYNCML:SYNCML1.2'>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>2</SessionID>
      <!--Session id is NOT the same as the previous interrupted session-->
      <MsgID>1</MsgID>
      ...
   </SyncHdr>
   <SyncBody>
      <Alert>
         <CmdID>1</CmdID>
         <Data>225</Data> <!-- Alert code  to 'Resume  Session' -->
         <Item>
            <Target>
               <LocURI>./Contact/Contacts</LocURI>
            </Target>
            <Source>
               <LocURI>./C\System\Data\Contacts.cdb</LocURI>
            </Source>
            <Meta>
               <Anchor xmlns='syncml:metinf'>
                  <Next>20021101T124234Z</Next> <!-- Updated -->
               </Anchor>
            </Meta>
         </Item>
      </Alert>
      ...
      <Final/>
   </SyncBody>
</SyncML>
```

The alert code for 'Resume Sync Session' does not contain the sync type of previous session, which is necessary to know prior to sending and receiving the <Sync> commands. When the client requires a resumption to the server, the latter can

choose to either accept the request or to reject it - in order to force a refresh sync. If the server accepts the request, it MUST use the same sync type as for the interrupted session. If it rejects the request, it MUST reply with the status code 508.
If the status code for 'Resume Sync Session' is a 200 'Success', alert for sync type in server's response is used by the server to indicate the type of sync being resumed. If the status code for 'Resume Sync Session' is 508 'Refresh Required' then alert for sync type is used by the server to indicate the new sync type.

The following code snippets elaborate how the server accepts or rejects the resume alert and based on that indicates the relevant sync-type, which is either the previous session's sync-type or forces a refresh sync.

```
<?xml version="1.0" encoding="UTF-8"?>
<SyncML xmlns='SYNCML:SYNCML1.2'>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>2</SessionID>
      <MsgID>1</MsgID>
      ...
   </SyncHdr>
   <SyncBody>
      ...
      <Status>
         <CmdID>2</CmdID>
         <MsgRef>1</MsgRef>
         <CmdRef>1</CmdRef>
         <Cmd>Alert</Cmd>
         <TargetRef>./.Contact/Contacts</TargetRef>
         <SourceRef>./C\System\Data\Contacts.cdb</SourceRef>
         <Data>200</Data> <!--Server Accepts to Resume previous Sync Session
               OR Server Rejects to Resume, instead forces a Refresh, status 508-->
         <Item>
            <Data>
               <Anchor xmlns='syncml:metinf'>
                  <Next>20021101T124234Z</Next>
               </Anchor>
            </Data>
         </Item>
      </Status>
      <Alert>
         <CmdID>3</CmdID>
         <Data>201</Data>  <!—Server uses the previous Sync type OR -->
            <!—Server uses 201, which is not similar to the resuming previous slow-sync
            session. The difference being that server has rejected to Resume the previous
            slow-sync (status to alert for resume MUST be 508) and has forced a 'Refresh'
            i.e. a full slow-sync, instead.-->
         <Item>
            <Target>
               <LocURI>./C\System\Data\Contacts.cdb</LocURI>
            </Target>
            <Source>
               <LocURI>./Contact/Contacts</LocURI>
            </Source>
            <Meta>
               <Anchor xmlns='syncml:metinf'>
                  <Last>1</Last> <!-- Not updated -->
                  <Next>2</Next> <!-- Updated -->
               </Anchor>
            </Meta>
         </Item>
      </Alert>
      ...
      <Final/>
   </SyncBody>
</SyncML>
```

6.14 Busy Signaling

If the server is able to receive the data from the client but it is not able to process the request(s) at a reasonable time[1] after receiving the modifications from the client, the server SHOULD send information about that to the client. This happens by sending the Busy Status package back to the client.

After the client has received a busy signal from the server, the client MAY ask for the sync results later or start the synchronization from the beginning. If the client starts the synchronization from the beginning its 'Last' sync anchor MUST NOT be updated.

If the server has sent the busy status to the client and it does not get a request from the client (i.e., Result Alert), the server MUST assume that the client has stopped the synchronization and start the synchronization from the beginning. The server MUST NOT update its 'Last' sync nor the client 'Next' sync anchors.

6.14.1 Busy Status from Server

Informing the client that the server is busy happens by sending the Busy Status package to the client. This can be sent before any package is completely received. The Busy Status package MUST NOT be used to return status information related to any individual data items or command which are in SyncBody of the client request.

The requirements for the elements within the Busy Status package are:

1. 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 'SyncML/1.2'.
   - Session ID MUST be included to indicate the ID of a sync session.
   - MsgID MUST be used to unambiguously identify the message belonging to a sync session and traveling from the server to the client.
   - The Target element MUST be used to identify the target device.
   - The Source element MUST be used to identify the source device and service.
2. The Status element for the SyncHdr MUST be included in SyncBody.
   - The status code (101, in progress) MUST be returned within the Status for the command sent by the client. The status is returned for the SyncHdr command.
3. The Final element MUST NOT be used for the message.

6.14.1.1 Example of Busy Status

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
      <Target><LocURI>IMEI:493005100592800</LocURI></Target>
```

[1] This time is dependent e.g. on the transport protocol transferring SyncML messages.

```
      <Source><LocURI>http://www.syncml.org/sync-server</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>2</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
         <TargetRef>http://www.syncml.org/sync-server</TargetRef>
         <SourceRef>IMEI:493005100592800</SourceRef>
         <Data>101</Data> <!--Statuscode for Busy-->
      </Status>
   </SyncBody>
</SyncML>
```

6.14.2 Result Alert from Client

The result alert is sent to ask results to the last message which was sent to the server. This is done by sending a Result Alert package from the client to the server. A message within this package has the following requirements.

1. 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 'SyncML/1.2'.
   - Session ID MUST be included to indicate the ID of a sync session.
   - MsgID MUST be used to unambiguously identify the message belonging to a sync session and traveling from the client to the server.
   - The Target element MUST be used to identify the target device and service.
   - The Source element MUST be used to identify the source device.
2. The Alert element MUST be included in SyncBody. There are the following requirements for this Alert element.
   - CmdID MUST be used.
   - The Item element is used to specify the server and the client device.
   - The Data element is used to include the Alert code. The alert code is '221' (See **Alert Codes**).
3. The Final element MUST NOT be used for the message.

If the server is still busy, when it receives this Result Alert from the client, it MUST again return the Busy Status with the '101' status code back to client. The status code is associated with the SyncHdr and the Alert command sent by the client.

6.14.2.1 Example of Result Alert

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>3</MsgID>
      <Target><LocURI>http://www.syncml.org/sync-server</LocURI></Target>
      <Source><LocURI>IMEI:493005100592800</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Alert>
         <CmdID>1</CmdID>
         <Data>221</Data>
         <Item>
            <Target><LocURI>http://www.syncml.org/sync-server</LocURI></Target>
```

```
          <Source><LocURI>IMEI:493005100592800</LocURI></Source>
        </Item>
      </Alert>
   </SyncBody>
</SyncML>
```

7. Authentication

In this chapter, the authentication procedures are defined for the basic and MD5 digest access authentication. Both of them MUST be supported by the devices conforming to this specification.

7.1 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 [DSREPU]). The Chal contains a challenge applicable to the requested resource. The device MAY repeat the request with a suitable Cred element (See [DSREPU]). If the request already included the Cred element, then the 401 response indicates that authorization has been refused for those credentials.

Both, the sync client and the sync 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 synchronization 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 sync 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 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 server layer (see chapter 7.3) contains only Status commands (i.e. Put, 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 than the previous messages and the message MUST be sent to the RespURI, if it was specified in the response indicating the authentication failure.

7.2 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 [DSREPU]. The authentication type is dependent on the challenge (See the previous chapter) or the pre-configuration.

7.3 Server Layer Authentication

When the authentication is considered, this protocol mandates only the support for the authentication on the server layer (in the SyncHdr element). I.e., the authentication of the server layer MUST be supported by the device complying with this specification.

The authentication on the server 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 sync client can authenticate itself to the sync server and the sync server can authenticate itself to the client.

7.4 Authentication of Database Layer

The authentication of the database layer SHOULD be supported by the device complying with this specification. The authentication on the database layer is accomplished by using the Cred element in the Alert and Sync commands (See the [DSREPU].) and the Status command associated with these commands. Within the Status command, the challenge for the authentication is carried as defined earlier. The authentication can happen both directions, i.e., the sync client can authenticate itself to the sync server and the sync server can authenticate itself to the client (Alert and Sync command are sent both directions).

7.5 Authentication Examples

7.5.1 Basic authentication with a challenge

At this example, the client tries to initiate sync with the server without any credentials (Pkg #1). The server challenges the client (Pkg #2) for the server 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>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>1</MsgID>
      <Target><LocURI>http://www.syncml.org/sync-server</LocURI></Target>
      <Source><LocURI>IMEI:493005100592800</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      ...
   </SyncBody>
</SyncML>
```

Pkg #2 from Server

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>1</MsgID>
      <Target><LocURI>IMEI:493005100592800</LocURI></Target>
      <Source><LocURI>http://www.syncml.org/sync-server</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>1</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
         <TargetRef>http://www.syncml.org/sync-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>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
      <Target><LocURI>http://www.syncml.org/sync-server</LocURI></Target>
      <Source><LocURI>IMEI:493005100592800</LocURI></Source>
      <Cred>
         <Meta><Type xmlns='syncml:metinf'>syncml:auth-basic</Type></Meta>
         <Data>QnJ1Y2UyOk9oQmVoYXZl</Data> <!—base64 formatting of "userid:password"-->
      </Cred>
   </SyncHdr>
   <SyncBody>
      ...
   </SyncBody>
</SyncML>
```

Pkg #2 from Server

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
      <Target><LocURI>IMEI:493005100592800</LocURI></Target>
      <Source><LocURI>http://www.syncml.org/sync-server</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>2</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
         <TargetRef>http://www.syncml.org/sync-server</TargetRef>
         <SourceRef>IMEI:493005100592800</SourceRef>
         <Data>212</Data>
         <!--Authenticated for session-->
      </Status>
      ...
   </SyncBody>
</SyncML>
```

At this example, the client initiates the synchronization by sending credentials with the Pkg#1, and server may include its credentials into the Pkg#2.

Pkg #1 (with credentials) from Client

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
```

```
      <Target><LocURI>http://www.syncml.org/sync-server</LocURI></Target>
      <Source><LocURI>IMEI:493005100592800</LocURI></Source>
      <Cred>
         <Meta><Type xmlns='syncml:metinf'>syncml:auth-basic</Type></Meta>
         <Data>QnJ1Y2UyOk9oQmVoYXZl</Data>
         <!—base64 formatting of "userid:password"-->
      </Cred>
   </SyncHdr>
   <SyncBody>
      ...
   </SyncBody>
</SyncML>
```

Pkg #2 (with credentials) from Server

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
      <Target><LocURI>IMEI:493005100592800</LocURI></Target>
      <Source><LocURI>http://www.syncml.org/sync-server</LocURI></Source>
      <Cred>
         <Meta><Type xmlns='syncml:metinf'>syncml:auth-basic</Type></Meta>
         <Data>QnJ1Y2UyOk9oQmVoYXZl</Data> <!—base64 formatting of "userid:password"-->
      </Cred>
   </SyncHdr>
   <SyncBody>
   <Status>
      <CmdID>1</CmdID>
      <MsgRef>1</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
      <TargetRef>http://www.syncml.org/sync-server</TargetRef>
      <SourceRef>IMEI:493005100592800</SourceRef>
      <Data>212</Data>
      <!--Authenticated for session-->
   </Status>
   ...
   </SyncBody>
</SyncML>
```

7.5.2 MD5 digest access authentication with a challenge

At this example, the client tries to initiate sync with the server without any credentials (Pkg #1). The server challenges the client (Pkg #2) for the server layer authentication. The authentication type I is now the MD5 digest access authentication. The client MUST send Pkg #1 again with the credentials. The server accepts the credentials and the session is authenticated (Pkg #2). Also, the server sends the next nonce to the client, which the client MUST use when the next sync session is started. In the example, commands in SyncBody are not shown although in practice, they would be there.

Pkg #1 from Client

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>1</MsgID>
      <Target><LocURI>http://www.syncml.org/sync-server</LocURI></Target>
      <Source>
         <LocURI>IMEI:493005100592800</LocURI>
      </Source>
   </SyncHdr>
   <SyncBody>
```

```
      ...
   </SyncBody>
</SyncML>
```

Pkg #2 from Server

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>1</MsgID>
      <Target><LocURI>IMEI:493005100592800</LocURI></Target>
      <Source><LocURI>http://www.syncml.org/sync-server</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>2</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
         <TargetRef>http://www.syncml.org/sync-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'>Tm9uY2U=</NextNonce>
            </Meta>
         </Chal>
         <Data>407</Data>
         <!--Credentials missing-->
      </Status>
         ...
   </SyncBody>
</SyncML>
```

Pkg #1 (with credentials) from Client

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

Pkg #2 from Server

```
<SyncML>
   <SyncHdr>
```

```
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
      <Target><LocURI>IMEI:493005100592800</LocURI></Target>
      <Source><LocURI>http://www.syncml.org/sync-server</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>1</MsgRef>
         <CmdRef>0</CmdRef>
         <Cmd>SyncHdr</Cmd>
         <TargetRef>http://www.syncml.org/sync-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>
```

8. Sync Initialization

The sync initialization implies that the actual synchronization (See Chapters 9-11), i.e., the sync commands, can also be transmitted and processed. Prior to the sync initialization, the OMA DS server might alert the client to trigger synchronization with it (See Chapter 12) but this does not remove the need for the initialization. The sync initialization has the following purposes:

- To process the authentication between the client and the server on the SyncML level.
- To indicate which databases could be synchronized and which protocol type could be used.
- To enable the exchange of service and device capabilities.

The two first ones are done by using the Alert command of the SyncML Representation protocol. These MUST be supported by the client and the server.

The exchange of service capabilities is done by utilizing the Put and Get commands of the SyncML Representation protocol and the Device Information DTD (See also Chapter 6.7).

The initialization procedure is depicted in the figure below. Some parts of the procedure (some responses) can be included in the actual synchronization messages if it is necessary.

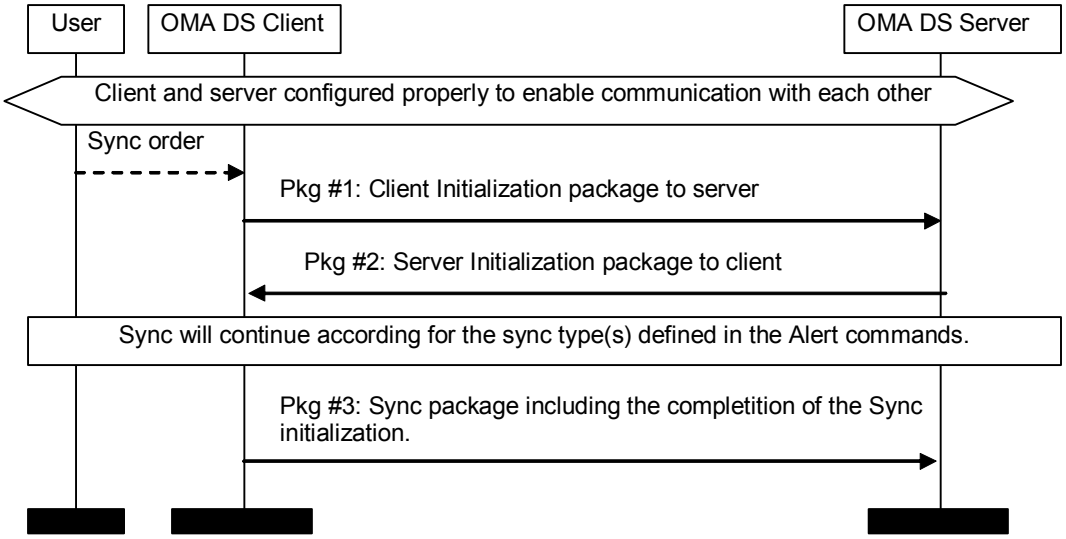

Figure 10 MSC of Synchronization Initialization

The arrows in all figures in this document represent SyncML packages, which can include one or more messages. The package flow presented above is one OMA DS session that means that all messages have the same OMA DS session ID.

The purpose and the requirements for each of the packages in the figure above are considered in the next sections.

8.1 Initialization Requirements for Client

As described in the previous chapter, the client needs to inform the server which databases it wants to synchronize and which type synchronization is desired. Optionally, the client can also include the authentication information and the service capabilities information into this initialization.

The databases, which are desired to be synchronized, are indicated in the separate Alert commands. I.e., for each database, a separate Alert command MUST be included in the SyncBody. In addition, the Alert command is used to exchange the sync anchors.

The synchronization type is indicated in the Alert command. See the alert codes in **Alert Codes**.

The authentication information, if it is included, MUST be placed inside the Cred element in the SyncHdr. Either the Basic or the MD5 Digest credential type can be used.

The service capabilities can be sent by using the Put command in the SyncBody element. The client MUST include service and device information, which is applicable from the Device Information DTD, in the data to be sent to the server. The client can also ask the service capabilities of the server. The Get command is used for this operation.

The detailed requirements for the sync initialization package (Pkg #1 in Figure 10) from the client to the server are:

1. Requirements for the elements within the SyncHdr element.
   - The value of the VerDTD element MUST be '1.2'.
   - The VerProto element MUST be included to specify the sync protocol and the version of the protocol. The value MUST be 'SyncML/1.2' when complying with this specification.
   - Session ID MUST be included to indicate the ID of a sync session.
   - MsgID MUST be used to unambiguously identify the message belonging to a sync session and traveling from the client to the server.
   - The Target element MUST be used to identify the target device and service.
   - The Source element MUST be used to identify the source device.
   - The Cred element MUST be included if the authentication is needed.
2. The Alert element(s) for each database to be synchronized MUST be included in SyncBody and the following requirements exist.
   - CmdID MUST be used.
   - NoResp SHOULD NOT be specified with the Alert command.
   - The Data element is used to include the Alert code. The alert code is one of the codes used at the initialization. See the alert codes in **Alert Codes**.
   - Target in the Item element is used to specify the target database.
   - Source in the Item element is used to specify the source database.
   - The sync anchors of the client MUST be included to specify the previous and current (Last and Next) sync anchors (See also Chapter 6.2.1). The sync anchors are carried inside the Meta element in the Item element.
3. If the service capabilities are sent from the client to the server, the following requirements for the Put command in the SyncBody exist.
   - CmdID MUST be used.
   - The Type element of the MetaInf DTD MUST be included in the Meta element of the Put command to indicate that the type of the data is the type of the Device Information DTD.
   - The Source element in the Item element MUST have a value './devinf12'.
   - The Data element is used to carry the device and service information data.
4. If the service capabilities are requested from the server, the following requirements for the Get command in the SyncBody exist.
   - CmdID MUST be used.

- The Type element of the MetaInf DTD MUST be included in the Meta element of the Get command to indicate that the type of the data is the type of the Device Information DTD.
- The Target element in the Item element MUST have a value ‘./devinf12’.

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

8.1.1 Example of Sync Initialization Package from Client

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>4</SessionID>
      <MsgID>1</MsgID>
      <Target><LocURI>http://www.syncml.org/sync-server</LocURI></Target>
      <Source><LocURI>IMEI:493005100592800</LocURI></Source>
      <Cred>
         <!--The authentication is optional.-->
         <Meta><Type xmlns='syncml:metinf'>syncml:auth-basic</Type></Meta>
         <Data>QnJ1Y2UyOk9oQmVoYXZl</Data>
         <!--base64 formatting of "userid:password"-->
      </Cred>
      <Meta>
         <!-- The Meta is now used to indicate the maximum SyncML message size -->
         <!-- that the client can receive. -->
         <MaxMsgSize xmlns='syncml:metinf'>5000</MaxMsgSize>
      </Meta>
   </SyncHdr>
   <SyncBody>
      <Alert>
         <CmdID>1</CmdID>
         <Data>200</Data>  <!-- 200 = TWO_WAY_ALERT -->
         <Item>
            <Target><LocURI>./contacts/james_bond</LocURI></Target>
            <Source><LocURI>./dev-contacts</LocURI></Source>
            <Meta>
               <Anchor xmlns='syncml:metinf'>
         <Last>234</Last>
         <Next>276</Next>
               </Anchor>
            </Meta>
         </Item>
      </Alert>
      <Put>
         <CmdID>2</CmdID>
         <Meta>
            <Type xmlns='syncml:metinf'>application/vnd.syncml-devinf+xml</Type>
         </Meta>
         <Item>
            <Source><LocURI>./devinf12</LocURI></Source>
            <Data>
               <DevInf xmlns='syncml:devinf'>
         <Man>Big Factory, Ltd.</Man>
         <Mod>4119</Mod>
         <OEM>Jane's phones</OEM>
         <FwV>2.0e</FwV>
         <SwV>2.0</SwV>
         <HwV>1.22I</HwV>
```

```
<DevId>1218182THD000001-2</DevId>
<DevTyp>phone</DevTyp>
<UTC/>
        <SupportLargeObjects/>
<SupportNumberOfChanges/>
<DataStore>
   <SourceRef>./contacts</SourceRef>
   <DisplayName>Phonebook</DisplayName>
   <MaxGUIDSize>32</MaxGUIDSize>
   <Rx-Pref>
      <CTType>text/x-vcard</CTType>
      <VerCT>2.1</VerCT>
   </Rx-Pref>
   <Tx-Pref>
      <CTType>text/x-vcard</CTType>
      <VerCT>2.1</VerCT>
   </Tx-Pref>
   <CTCap>
      <CTType>text/x-vcard</CTType>
      <VerCT>2.1</VerCT>
      <Property>
         <PropName>BEGIN</PropName>
         <ValEnum>VCARD</ValEnum>
      </Property>
      <Property>
         <PropName>END</PropName>
         <ValEnum>VCARD</ValEnum>
      </Property>
      <Property>
         <PropName>VERSION</PropName>
         <ValEnum>2.1</ValEnum>
      </Property>
      <Property>
         <PropName>N</PropName>
      </Property>
      <Property>
         <PropName>TEL</PropName>
         <MaxOccur>5</MaxOccur>
         <PropParam>
            <ParamName>TYPE</ParamName>
            <ValEnum>VOICE, CELL</ValEnum>
            <ValEnum>VOICE, HOME</ValEnum>
         </PropParam>
      </Property>
      <Property>
         <PropName>NOTE</PropName>
         <MaxOccur>1</MaxOccur>
         <MaxSize>255</MaxSize>
         <NoTruncate/>
      </Property>
      <Property>
         <PropName>PHOTO</PropName>
         <MaxOccur>1</MaxOccur>
         <PropParam>
            <ParamName>TYPE</ParamName>
            <ValEnum>JPEG</ValEnum>
         </PropParam>
      </Property>
   </CTCap>
   <SyncCap>
      <SyncType>01</SyncType>
      <SyncType>02</SyncType>
      <SyncType>07</SyncType>
   </SyncCap>
```

```
        </DataStore>
             </DevInf>
          </Data>
        </Item>
      </Put>
      <Get>
        <CmdID>3</CmdID>
        <Meta><Type xmlns='syncml:metinf'>application/vnd.syncml-devinf+xml</Type></Meta>
        <Item>
          <Target><LocURI>./devinf12</LocURI></Target>
        </Item>
      </Get>
      <Final/>
   </SyncBody>
</SyncML>
```

8.2 Initialization Requirements for Server

When the server has received the Initialization package from the client, it completes the initialization phase by responding to the client from the server perspective. To complete the initialization, the server sends its authentication information, sync anchors, and device information back to the client. Also, the server MUST accept the sync type.

The detailed requirements for the sync initialization package (Pkg #2 in Figure 4) from the server to the client are:

1. Requirements for the elements within the SyncHdr element.
   - The value of the VerDTD element MUST be '1.2'.
   - The VerProto element MUST be included to specify the sync protocol and the version of the protocol. The value MUST be 'SyncML/1.2' when complying with this specification.
   - Session ID MUST be included to indicate the ID of a sync session.
   - MsgID MUST be used to unambiguously identify the message belonging to a sync session and traveling from the server to the client.
   - The Target element MUST be used to identify the target device and service.
   - The Source element MUST be used to identify the source device.
   - The Cred element MUST be included if the authentication is needed.
2. The Status MUST be returned for the Alert command sent by the client if the client requested the response. This can be sent before Package #1 is completely received (See Chapter 6.9).
   - If the client is not authenticated to use the service, the sync type is wrong (e.g., slow sync needed), or some other error occurs, the server MUST return an error for that.
   - The next sync anchor of the client MUST be included in the Data element of Item (See 6.2.1).
3. If the client sent the device information to the server, the server MUST be able to retrieve them and the Status MUST be returned for that command. This can be sent before Package #1 is completely received.
4. If the client requested the device information of the server, the Results element MUST be returned. This can be sent before Package #1 is completely received.
   - The Type element of the MetaInf DTD MUST be included in the Meta element in the Results element to indicate that the type of the data is the type of the Device Information DTD.
   - The Source element in the Item element MUST have a value './devinf12'.
   - The Data element is used to carry the device and service information of the server.

5. The Alert element(s) for each database to be synchronized MUST be included in SyncBody and the following requirements exist.
   - CmdID MUST be used.
   - NoResp SHOULD NOT be specified with the Sync command.
   - The Data element is used to include the alert code. If this is different that the alert code sent by the client, the client SHOULD follow this when synchronization is continued.
   - Target is used to specify the target database.
   - Source is used to specify the source database.
   - The sync anchors of the server MUST be included to specify the previous and current (Last and Next) sync anchors of the server (See also Chapter 6.2.1).
6. If the service capabilities were not asked by the client, the server MAY send them to the client by using the Put command. The following requirements for the Put command in the SyncBody exist.
   - CmdID MUST be used.
   - The Type element of the MetaInf DTD MUST be included in the Meta element of the Put command to indicate that the type of the data is the type of the Device Information DTD.
   - The Source element in the Item element MUST have a value ‘./devinf12’.
   - The Data element is used to carry the device and service information data of the server.
7. If the client did not send its service capabilities and the server needs to receive them, the server can request those by using the Get command. The following requirements for the Get command in the SyncBody exist.
   - CmdID MUST be used.
   - The Type element of the MetaInf DTD MUST be included in the Meta element of the Get command to indicate that the type of the data is the type of the Device Information DTD.
   - The Target element in the Item element MUST have a value ‘./devinf12’.
8. The Final element MUST be used for the message, which is the last in this package.

To complete the sync initialization from the client side, the client MUST respond to the commands (Alert, possible Put and Get) sent by the server. The Status elements and the Result element associated with the commands can be returned in the first package occurring in actual synchronization (Refer Package #3 in Two-way synchronization and One-way synchronizations.

8.2.1 Example of Sync Initialization Package from Server

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>4</SessionID>
      <MsgID>1</MsgID>
      <Target><LocURI>IMEI:493005100592800</LocURI></Target>
      <Source><LocURI>http://www.syncml.org/sync-server</LocURI></Source>
      <Cred> <!--The authentication is optional.-->
         <Meta><Type xmlns='syncml:metinf'>syncml:auth-basic</Type></Meta>
         <Data>QnJ1Y2UyOk9oQmVoYXZl</Data> <!--base64 formatting of "userid:password"-->
      </Cred>
   </SyncHdr>
   <SyncBody>
      <Status>
```

```
   <CmdID>1</CmdID>
   <MsgRef>1</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
   <TargetRef>http://www.syncml.org/sync-server</TargetRef>
   <SourceRef>IMEI:493005100592800</SourceRef>
   <Data>212</Data> <!--Statuscode for OK, authenticated for session-->
</Status>
<Status>
   <CmdID>2</CmdID>
   <MsgRef>1</MsgRef>
   <CmdRef>1</CmdRef>
   <Cmd>Alert</Cmd>
   <TargetRef>./contacts/james_bond</TargetRef>
   <SourceRef>./dev-contacts</SourceRef>
   <Data>200</Data> <!--Statuscode for OK-->
   <Item>
      <Data>
         <Anchor xmlns='syncml:metinf'>
            <Next>276</Next>
         </Anchor>
      </Data>
   </Item>
</Status>
<Status>
   <CmdID>3</CmdID>
   <MsgRef>1</MsgRef>
   <CmdRef>2</CmdRef>
   <Cmd>Put</Cmd>
   <SourceRef>./devinf12</SourceRef>
   <Data>200</Data> <!--Statuscode for OK-->
</Status>
<Results>
   <CmdID>4</CmdID>
   <MsgRef>1</MsgRef><CmdRef>3</CmdRef>
   <Meta><Type xmlns='syncml:metinf'>application/vnd.syncml-devinf+xml</Type></Meta>
   <Item>
      <Source><LocURI>./devinf12</LocURI></Source>
      <Data>
         <DevInf xmlns='syncml:devinf'>
            <Man>Small Factory, Ltd.</Man>
            <Mod>Tiny Server</Mod>
            <OEM>Tiny Shop</OEM>
            <DevId>485749KR</DevId>
            <DevTyp>Server</DevTyp>
            <UTC/>
            <SupportLargeObjects/>
            <SupportNumberOfChanges/>
            <DataStore>
               <SourceRef>./contacts</SourceRef>
               <DisplayName>Addressbook</DisplayName>
               <Rx-Pref>
                  <CTType>text/x-vcard</CTType>
                  <VerCT>2.1</VerCT>
               </Rx-Pref>
               <Rx>
                  <CTType>text/vcard </CTType>
                  <VerCT>3.0</VerCT>
               </Rx>
               <Tx-Pref>
                  <CTType>text/x-vcard</CTType>
                  <VerCT>2.1</VerCT>
               </Tx-Pref>
               <Tx>
                  <CTType>text/vcard</CTType>
                  <VerCT>3.0</VerCT>
```

```
</Tx>
<CTCap>
   <CTType>text/x-vcard</CTType>
   <VerCT>2.1</VerCT>
   <Property>
      <PropName>BEGIN</PropName>
      <ValEnum>VCARD</ValEnum>
   </Property>
   <Property>
      <PropName>END</PropName>
      <ValEnum>VCARD</ValEnum>
   </Property>
   <Property>
      <PropName>VERSION</PropName>
      <ValEnum>2.1</ValEnum>
   </Property>
   <Property>
      <PropName>N</PropName>
   </Property>
   <Property>
      <PropName>TEL</PropName>
      <MaxOccur>8</MaxOccur>
      <PropParam>
         <ParamName>TYPE</ParamName>
         <ValEnum>VOICE, CELL</ValEnum>
         <ValEnum>VOICE, HOME</ValEnum>
         <ValEnum>FAX, HOME</ValEnum>
      </PropParam>
   </Property>
   <Property>
      <PropName>NOTE</PropName>
      <MaxOccur>1</MaxOccur>
      <MaxSize>1024</MaxSize>
      <NoTruncate/>
   </Property>
   <Property>
      <PropName>PHOTO</PropName>
      <MaxOccur>1</MaxOccur>
      <PropParam>
         <ParamName>TYPE</ParamName>
         <ValEnum>JPEG</ValEnum>
         <ValEnum>GIF</ValEnum>
      </PropParam>
   </Property>
</CTCap>
<CTCap>
   <CTType>text/vcard</CTType>
   <VerCT>3.0</VerCT>
   <Property>
      <PropName>BEGIN</PropName>
      <ValEnum>VCARD</ValEnum>
   </Property>
   <Property>
      <PropName>END</PropName>
      <ValEnum>VCARD</ValEnum>
   </Property>
   <Property>
      <PropName>VERSION</PropName>
      <ValEnum>3.0</ValEnum>
   </Property>
   <Property>
      <PropName>N</PropName>
   </Property>
   <Property>
```

```
                        <PropName>TEL</PropName>
                        <MaxOccur>8</MaxOccur>
                        <PropParam>
                           <ParamName>TYPE</ParamName>
                           <ValEnum>VOICE, CELL</ValEnum>
                           <ValEnum>VOICE, HOME</ValEnum>
                           <ValEnum>FAX, HOME</ValEnum>
                        </PropParam>
                     </Property>
                     <Property>
                        <PropName>NOTE</PropName>
                        <MaxOccur>1</MaxOccur>
                        <MaxSize>1024</MaxSize>
                        <NoTruncate/>
                     </Property>
                     <Property>
                        <PropName>PHOTO</PropName>
                        <MaxOccur>1</MaxOccur>
                        <PropParam>
                           <ParamName>TYPE</ParamName>
                           <ValEnum>JPEG</ValEnum>
                           <ValEnum>GIF</ValEnum>
                        </PropParam>
                     </Property>
                  </CTCap>
                  <SyncCap>
                     <SyncType>01</SyncType>
                     <SyncType>02</SyncType>
                     <SyncType>03</SyncType>
                     <SyncType>04</SyncType>
                     <SyncType>05</SyncType>
                     <SyncType>06</SyncType>
                     <SyncType>07</SyncType>
                  </SyncCap>
               </DataStore>
            </DevInf>
         </Data>
      </Item>
   </Results>
   <Alert>
      <CmdID>5</CmdID>
      <Data>200</Data> <!-- 200 = TWO_WAY_ALERT -->
      <Item>
         <Target><LocURI>./dev-contacts</LocURI></Target>
         <Source><LocURI>./contacts/james_bond</LocURI></Source>
         <Meta>
            <Anchor xmlns='syncml:metinf'>
               <Last>20040119T081812Z </Last>
               <Next>20040120T093223Z </Next>
            </Anchor>
         </Meta>
      </Item>
   </Alert>
   <Final/>
 </SyncBody>
</SyncML>
```

8.3 Error Case Behaviors

In this chapter, the recommended behaviors are defined in the cases of different error types, which can occur during the sync initialization.

8.3.1 No Packages from Server

If the client has sent its sync initialization package to the server and it does not get any complete response to it, the client MUST assume that the server has not received the sync initialization package of the client. The client MUST send its sync initialization package again later.

8.3.2 No Initialization Completion from Client

If the server has sent its sync initialization package to the client and it does not get any complete response to it (Refer Pkg #3), the server MUST assume that the client has not received the sync initialization package of the server. The server can drop the session and the sync initialization MUST be started from the beginning when synchronization is started at the next time.

8.3.3 Initialization Failure

If the initialization fails, a defined error code [DSREPU] is sent, the devices MUST act according that error type.

9. Two-Way Sync

Two-way sync is a normal synchronization type in which the client and the server exchange information about the modified data in these devices. The client is always the device which first sends the modifications. According to the information from the client, the server processes the synchronization request and the data from the client is compared and unified with the data in the server. After that, the server sends its modified data to the client device, which is then able to update its database with the data from the server.

In Figure 11, there is depicted the MSC of the client initiated two-way sync scenario.

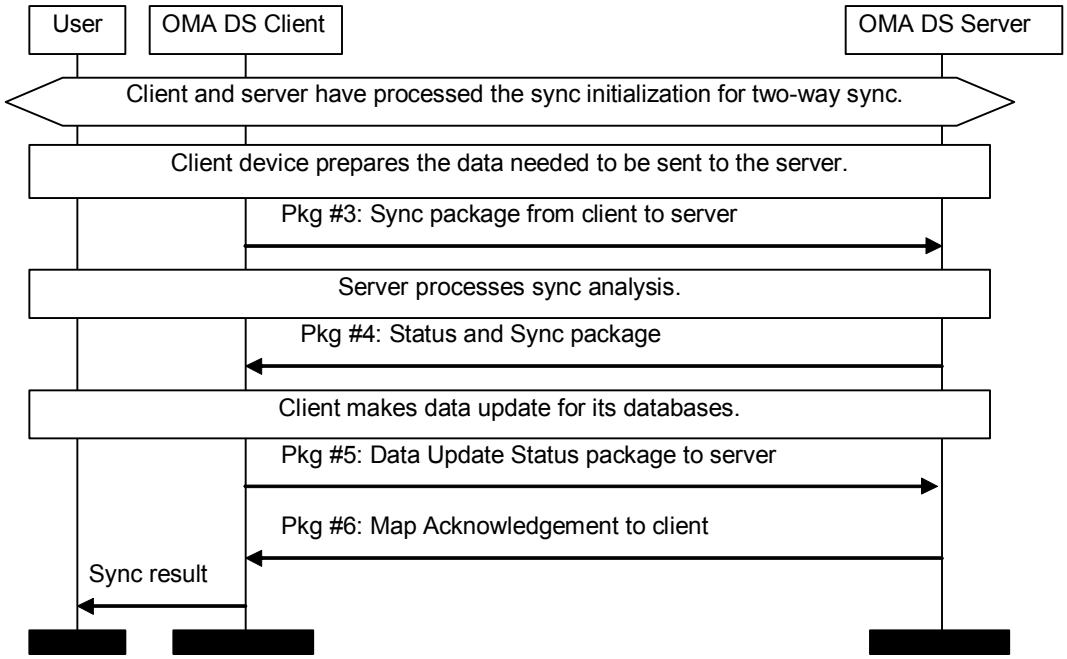

Figure 11 MSC of Two-Way Sync

The arrows in all figures in this document represent SyncML packages, which can include one or more messages. The package flow presented above is one OMA DS session that means that all messages have the same OMA DS session ID. The Session ID is same as used at the initialization.

The purpose and the requirements for each of the packages in the figure above are considered in the next sections.

Note. If the sync is done without a separate initialization (See Chapter 6.12) the number of a package in the figure does not describe the actual atomic number of a package in a synchronization session.

9.1 Client Modifications to Server

To enable sync, the client needs to inform the server about all client data modifications, which have happened since the previous sync package with modifications has been sent from the client to the server[2] (Refer to the sync package, Pkg #3 in Figure 11). Any client modification, which is done after sending this package, MUST be reported to the server during the next sync session. It is not allowed to put them inside subsequent packages from the client to the server. The requirements for the sync package from the client to the server are following.

[2] These modifications include also modifications which have happened during the previous sync session after the client has sent its modifications to the server.

1. Requirements for the elements within the SyncHdr element.
   - The value of the VerDTD element MUST be '1.2'.
   - The VerProto element MUST be included to specify the sync protocol and the version of the protocol. The value MUST be 'SyncML/1.2' when complying with this specification.
   - Session ID MUST be included to indicate the ID of a sync session.
   - MsgID MUST be used to unambiguously identify the message belonging to a sync session and traveling from the client to the server.
   - The Target element MUST be used to identify the target device and service.
   - The Source element MUST be used to identify the source device.
2. The Status MUST be returned for the Alert command sent by the server if status was requested by the server. This can be sent before Package #2 is completely received.
   - If the server is not authenticated to use the service, the sync type is wrong (e.g., slow sync needed), or some other error occurs, the client MUST return an error for that.
   - The next sync anchor of the server MUST be included in the Data element of Item (See 6.2.1).
3. If the server sent the device information to the client, the client SHOULD process the transmitted device information and the Status MUST be returned for that command if requested by the server. This can be sent before Package #2 is completely received.
4. If the server requested the device information of the client, the Results element MUST be returned. This can be sent before Package #2 is completely received.
   - The Type element of the MetaInf DTD MUST be included in the Meta element in the Results element to indicate that the type of the data is the type of the Device Information DTD.
   - The Source element in the Results element MUST have a value './devinf12'.
   - The Data element MUST be used to carry the device and service information of the client.
5. The Sync element MUST be included in SyncBody and the following requirements exist.
   - CmdID MUST be used.
   - NoResp SHOULD NOT be specified with the Sync command.
   - The Target element MUST be used to identify the target database.
   - The Source element MUST be used to identify the source database.
   - The free memory SHOULD be specified inside the Meta element. The free memory can be either the free memory amount in the source database or the free memory amount on the client device (See Chapter 6.8). If supplied this information MUST be sent in the first message belonging this package.
   - NumberOfChanges MAY be used to indicate the number of changes in the source database.
6. If there are modifications in the client, there are following requirements for the operational elements (e.g., Replace, Delete, and Add[3]) within the Sync element.
   - CmdID MUST be used.

[3] It is not required that the client uses the Add command when sending modifications. It MAY use the Replace command for additions, in which case the receiving device MUST interpret the command as and Add command.

- NoResp SHOULD NOT be specified with all these operations.
- The Source element MUST be included to indicate the LUID (See Definitions) of the data item within the Item element.
- The Type element of the MetaInf DTD MUST be included in the Meta element to indicate the type of the data item (E.g., MIME type). The Meta element inside an operation or inside an item can be used.
- Data element MUST be used to carry data itself if the operation is not a deletion.

7. The Final element MUST be used for the message, which is the last in this package. After the server has received the final message of the package, it can complete the sync analysis and send its modifications back to client.

9.1.1 Example of Sending Modifications to Server

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>4</SessionID>
      <MsgID>2</MsgID>
      <Target><LocURI>http://www.syncml.org/sync-server</LocURI></Target>
      <Source><LocURI>IMEI:493005100592800</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>1</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
         <TargetRef>IMEI:493005100592800</TargetRef>
         <SourceRef> http://www.syncml.org/sync-server</SourceRef>
         <Data>212</Data> <!--Statuscode for OK, authenticated for session-->
      </Status>
      <Status>
         <CmdID>2</CmdID>
         <MsgRef>1</MsgRef>
         <CmdRef>5</CmdRef>
         <Cmd>Alert</Cmd>
         <TargetRef>./dev-contacts</TargetRef>
         <SourceRef>./contacts/james_bond</SourceRef>
         <Data>200</Data> <!--Statuscode for Success-->
         <Item>
            <Data>
               <Anchor xmlns='syncml:metinf'>
         <Next>20040120T093223Z </Next>
               </Anchor>
            </Data>
         </Item>
      </Status>
      <Sync>
         <CmdID>3</CmdID>
         <Target><LocURI>./contacts/james_bond</LocURI></Target>
         <Source><LocURI>./dev-contacts</LocURI></Source>
         <Meta>
            <Mem xmlns='syncml:metinf'>
               <FreeMem>8100</FreeMem>
               <!--Free memory (bytes) in Calendar database on a device -->
               <FreeId>81</FreeId>
               <!--Number of free records in Calendar database-->
            </Mem>
         </Meta>
         <NumberOfChanges>1</NumberOfChanges>
         <Replace>
```

```
          <CmdID>4</CmdID>
          <Meta><Type xmlns='syncml:metinf'>text/x-vcard</Type></Meta>
          <Item>
             <Source><LocURI>1012</LocURI></Source>
             <Data><!--The vCard data would be placed here.--></Data>
          </Item>
        </Replace>
      </Sync>
      <Final/>
   </SyncBody>
</SyncML>
```

9.2 Server Modifications to Client

The sync package (Refer Pkg #4 in Figure 11) to the client has the following purposes:

- To inform the client about the results of sync analysis.
- To inform about all data modifications, which have happened in the server since the previous time when the server has sent the modifications to the client.

Any server modifications, which are done after sending this package, MUST be reported to the client during the next sync session. It is not allowed to put them inside subsequent packages from the server to the client.

The requirements for messages within this sync package are following.

1. 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 'SyncML/1.2'.
   - Session ID MUST be included to indicate the ID of a sync session.
   - MsgID MUST be used to unambiguously identify the message belonging to a sync session and traveling from the server to the client.
   - The Target element MUST be used to identify the target device.
   - The Source element MUST be used to identify the source device and service.
2. The Status element MUST be included in SyncBody if requested by the client. It is now used to indicate the general status of the sync analysis and the status information related to data items sent by the client (e.g., a conflict has happened.). Status information for data items can be sent before Package #3 is completely received.
3. The Sync element MUST be included in SyncBody, if earlier there were no occurred errors, which could prevent the server to process the sync analysis and to send its modifications back to the client. For the Sync element, there are the following requirements.
   - CmdID MUST be used.
   - The response MAY be REQUIRED for the Sync command. (See the Caching of Map Item, Chapter 6.3.1)
   - The Target element MUST be used to identify the target database.
   - The Source element MUST be used to identify the source database.
   - NumberOfChanges MUST be used to indicate the number of changes in the source database if the client has indicated that it supports NumberOfChanges.

4. If there is any modification in the server after the previous sync, there are following requirements for the operational elements (e.g., Replace, Delete, and Add[4]) within the Sync element.

   - CmdID MUST be used.
   - The response MAY be REQUIRED for these operations.
   - Source MUST be used to define the temporary GUID (See Definitions) of the data item in the server if the operation is an addition. If the operation is not an addition, Source MUST NOT be included.
   - Target MUST be used to define the LUID (See Definitions) of the data item if the operation is not an addition. If the operation is an addition, Target MUST NOT be included.
   - The Data element inside Item is used to include the data itself if the operation is not a deletion.
   - The Type element of the MetaInf DTD MUST be included in the Meta element to indicate the type of the data item (E.g., MIME type). The Meta element inside an operation or inside an item can be used.

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

9.2.1 Example of Sending Modifications to Client

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>4</SessionID>
      <MsgID>2</MsgID>
      <Target><LocURI>IMEI:493005100592800</LocURI></Target>
      <Source><LocURI>http://www.syncml.org/sync-server</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>2</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
         <TargetRef>http://www.syncml.org/sync-server</TargetRef>
         <SourceRef>IMEI:493005100592800</SourceRef>
         <Data>200</Data>
      </Status>
      <Status><!--This is a status for the client modifications to the server.-->
         <CmdID>2</CmdID>
         <MsgRef>2</MsgRef><CmdRef>3</CmdRef><Cmd>Sync</Cmd>
         <TargetRef>./contacts/james_bond</TargetRef>
         <SourceRef>./dev-contacts</SourceRef>
         <Data>200</Data> <!--Statuscode for Success-->
      </Status>
      <Status>
         <CmdID>3</CmdID>
         <MsgRef>2</MsgRef>
         <CmdRef>4</CmdRef>
         <Cmd>Replace</Cmd>
         <SourceRef>1012</SourceRef>
         <Data>200</Data> <!--Statuscode for Success-->
      </Status>
      <Sync>
```

It is not required that the server uses the Add command when sending modifications. It MAY use the Replace command for additions, in which case the receiving device MUST interpret the command as and Add command.

```
        <CmdID>4</CmdID>
        <Target><LocURI>./dev-contacts</LocURI></Target>
        <Source><LocURI>./contacts/james_bond</LocURI></Source>
        <NumberOfChanges>2</NumberOfChanges>
        <Replace>
           <CmdID>5</CmdID>
           <Meta><Type xmlns='syncml:metinf'>text/x-vcard</Type></Meta>
           <Item>
              <Target><LocURI>1023</LocURI></Target>
              <Data><!--The vCard data would be placed here.--></Data>
           </Item>
        </Replace>
        <Add>
           <CmdID>6</CmdID>
           <Meta><Type xmlns='syncml:metinf'>text/x-vcard</Type></Meta>
           <Item>
              <Source><LocURI>10536681</LocURI></Source>
              <Data><!--The vCard data would be placed here.--></Data>
           </Item>
        </Add>
     </Sync>
     <Final/>
  </SyncBody>
</SyncML>
```

9.3 Data Update Status from Client

The data update status package from the client to the server is needed to transport the information about the result of the data update on the client side. In addition, it is used to indicate the LUID's of the new data items, which have been added in the client, i.e., the Map operation for mapping LUID's and temporary GUID's is sent to the server.

Note. This package MAY NOT be sent if the server has indicated that it does not require a response to its last package to the client. If the client decides that it does not send this message, it MUST be able to cache the Map operations until the next synchronization will happen, when these Map operations can be sent to the server (See also Chapter 6.3.1). However, the client is always allowed to send this Data Update Status package to the server, even if the server has not requested a response.

The messages in this package have the following requirements.

1. 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 'SyncML/1.2'.
   - Session ID MUST be included to indicate the ID of a sync session.
   - MsgID MUST be used to unambiguously identify the message belonging to a sync session and traveling from the client to the server.
   - The Target element MUST be used to identify the target device and service.
   - The Source element MUST be used to identify the source device.
2. The Status element MUST be in SyncBody if requested by the server. It is used to indicate the results of data update in the client. Also, the status information related to the individual data items is transferred to the server. The status information for data items can be sent before Package #4 is completely received.
3. The Map element MUST be included in the SyncBody element if the client has processed any server additions to its database. For each database being synchronized, a separate Map operation or operations MUST be sent if any additions to a database is carried out. This command can be sent before Package #4 is completely received.
   - CmdID MUST be used.

- The Source and Target elements MUST be used in the Map element.
- The response MUST be used with the Map operation.
- The client MUST return the client side IDs, i.e., LUID's and the server side IDs (temporary GUID's) for the data items within MapItem elements.

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

9.3.1 Example of Data Update Status to Server

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>4</SessionID>
      <MsgID>3</MsgID>
      <Target><LocURI>http://www.syncml.org/sync-server</LocURI></Target>
      <Source><LocURI>IMEI:493005100592800</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>2</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
         <TargetRef>IMEI:493005100592800</TargetRef>
         <SourceRef> http://www.syncml.org/sync-server</SourceRef>
         <Data>200</Data>
      </Status>
      <Status>
         <CmdID>2</CmdID>
         <MsgRef>2</MsgRef><CmdRef>4</CmdRef><Cmd>Sync</Cmd>
         <TargetRef>./dev-contacts</TargetRef>
         <SourceRef>./contacts/james_bond</SourceRef>
         <Data>200</Data>
      </Status>
      <Status>
         <CmdID>3</CmdID>
         <MsgRef>2</MsgRef><CmdRef>5</CmdRef><Cmd>Replace</Cmd>
         <TargetRef>1023</TargetRef>
         <Data>200</Data>
      </Status>
      <Status>
         <CmdID>4</CmdID>
         <MsgRef>2</MsgRef><CmdRef>6</CmdRef><Cmd>Add</Cmd>
         <SourceRef>10536681</SourceRef>
         <Data>200</Data>
      </Status>
      <Map>
         <CmdID>5</CmdID>
         <Target><LocURI>./contacts/james_bond</LocURI></Target>
         <Source><LocURI>./dev-contacts</LocURI></Source>
         <MapItem>
            <Target><LocURI>10536681</LocURI></Target>
            <Source><LocURI>1024</LocURI></Source>
         </MapItem>
      </Map>
      <Final/>
   </SyncBody>
</SyncML>
```

9.4 Map Acknowledgement from Server

The Map Acknowledgement from the server to the client is needed to inform the client that the server has received the mapping information of the data items. This acknowledgement is sent back to the client even if there were no Map operations in last package from the client to the server.

The messages in this package have the following requirements.

1. 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 'SyncML/1.2'.
   - Session ID MUST be included to indicate the ID of a sync session.
   - MsgID MUST be used to unambiguously identify the message belonging to a sync session and traveling from the server to the client.
   - The Target element MUST be used to identify the target device.
   - The Source element MUST be used to identify the source device and service.
   -
2. The Status element(s) MUST be included in SyncBody. It is now used to indicate the status of the Map operation(s). This or these can be sent before Package #5 is completely received.
3. The Final element MUST be used for the message, which is the last in this package.

9.4.1 Example of Map Acknowledge

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>SyncML/1.2</VerProto>
      <SessionID>4</SessionID>
      <MsgID>3</MsgID>
      <Target><LocURI>IMEI:493005100592800</LocURI></Target>
      <Source><LocURI>http://www.syncml.org/sync-server</LocURI></Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>3</MsgRef><CmdRef>0</CmdRef><Cmd>SyncHdr</Cmd>
         <TargetRef>http://www.syncml.org/sync-server</TargetRef>
         <SourceRef>IMEI:493005100592800</SourceRef>
         <Data>200</Data>
      </Status>
      <Status>
         <CmdID>1</CmdID>
         <MsgRef>3</MsgRef><CmdRef>5</CmdRef><Cmd>Map</Cmd>
         <TargetRef>./contacts/james_bond </TargetRef>
         <SourceRef>./dev-contacts</SourceRef>
         <Data>200</Data>
      </Status>
      <Final/>
   </SyncBody>
</SyncML>
```

9.5 Slow Sync

The slow sync can be desired for many reasons, e.g., the client or the server has lost its change log information, the LUID's have wrapped around in the client, or the sync anchors mismatch. The slow sync is a form of the two-way synchronization in which all items in one or more databases are compared with each other on a field-by-field basis. In practice, the slow sync means that the client sends all its data in a database to the server and the server does the sync analysis (field-by-field) for this data and the data in the server. After the sync analysis, the server returns all needed modifications back to the client. Also, the client returns the Map items for all data items, which were added by the server.

Because of many reasons to process the slow sync, it can be either the client or the server, which indicates a need for this. If the client does this, it specifies in the Alert command that the sync type is the slow sync. The Alert command MAY be the same as at the sync initialization or the similar Alert command MAY be included when Package #3 is sent. The value of the Alert code is 201.

If there is a need for the server to initiate the slow sync, it happens by including the Alert operation with the 201 alert code. This alert operation MUST be the Alert operation at the Sync Initialization (Refer Package #2). After the client has received the status and the Alert operation for the slow sync, sync can be thought to start as if the client were initiating the slow sync in Package #3. However, the client MUST NOT include the Alert command anymore if it was the server, which alerted the slow sync.

If the client or the server needs to initiate the slow sync after receiving the alert for the normal synchronization, they need to send back an error status for that Alert in addition the slow sync alert. The error code, which is used in this case, MUST be 508 ('Refresh required'). If the client has not used a separate synchronization initialization, as specified in Chapter 6.12, it MUST send all updates in the next message to the server after receiving the error status and the Alert for a slow sync.

After the server has sent the Sync Alert, and if the client does not agree with the sync anchor in that Alert, then the Client MUST start a slow sync. This is done by sending back a Status on that Alert with 'Refresh Required' (508). In this same message, the client SHOULD start the slow sync. In this case, the client MUST NOT send another Alert to start the slow sync. Note that it is not necessary for the client to compare the sync anchor from the server.

If the devices are synchronizing with each other at the first time, the slow sync MUST be initiated.

9.6 Error Case Behaviors

In this chapter, the recommended behaviors are defined in the cases of different error types.

9.6.1 No Packages from Server after Initialization

If the client has sent its modifications to the server and it does not get the status associated with those modifications, the client MUST assume that the server has not received those client modifications. At the next time when synchronization is started, the modifications, to which the status was not received, MUST be sent to the server again.

9.6.2 No Data Update Status from Client

If the server has sent its modifications to the client and it does not get the status associated with those server modifications, the server MUST assume that the client has not received those server modifications. Thus, at the next time when synchronization is started, the server modifications in addition to new ones MUST be sent to the client.

9.6.3 No Data Map Acknowledge from Server

If the client has sent the Map operation(s) and it does not get any complete response to it, the client SHOULD assume that the server has not received the Map operation(s). Thus, the client SHOULD try to send the Map operation(s) again or at the next time when synchronization is started.

9.6.4 Errors with Defined Error Codes

If the device receives a defined error code [DSREPU], it MUST act according that error type.

10. One-Way Sync from Client Only

The one-way sync from the client only is the sync type in which the client sends all modifications to the server but the server does not send its modifications back to the client. Thus, after this type of sync, the server includes all modified data from the client but the client does not know about modifications in the server. In Figure 12, there is depicted the MSC for this scenario.

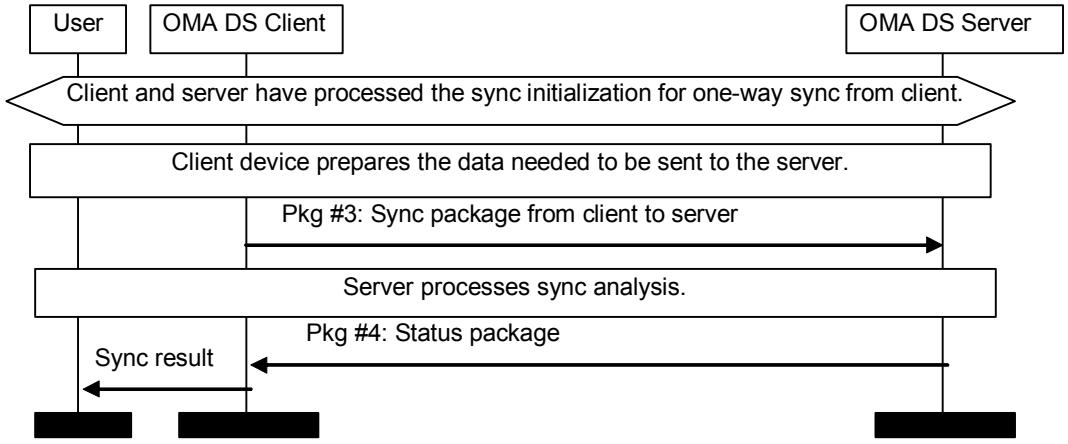

Figure 12 MSC of One-Way Sync from Client only

The package flow presented above is one OMA DS session that means that all messages have the same OMA DS session ID. The Session ID is same as used at the initialization.

The purpose and the requirements for each of package in the figure above are considered in the next sections.

Note. If the sync is done without a separate initialization (See Chapter 6.12), the number of a package in the figure does not describe the actual atomic number of a package in a synchronization session.

10.1 Client Modifications to Server

To initiate the sync, the client needs to inform the server about all client data modifications, which have happened since the previous sync[5] (Refer to the sync package, Pkg #3 in Figure 12). Any client modification, which is done after sending this package, MUST be reported to the server during the next sync session. It is not allowed to put them inside subsequent packages from the client to the server. The requirements for the sync package from the client to the server are the same as in Chapter 9.1.

10.2 Status from Server

The Status package (Refer Pkg #4) has a purpose of informing the client about the results of sync analysis. The requirements for the status package are following.

1. 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 'SyncML/1.2'.
    - Session ID MUST be included to indicate the ID of a sync session.

[5] These modifications include also modifications which have happened during the previous sync session after the client has sent its modifications to the server.

- MsgID MUST be used to unambiguously identify the message belonging to a sync session and traveling from the server to the client.
- Final MUST be used for the message, which is the last in this package.
- The Target element MUST be used to identify the target device.
- The Source element MUST be used to identify the source device and service.

2. The Status element MUST be included in SyncBody if requested by the client. It is now used to indicate the general status of the sync analysis and the status information related to data items sent by the client if this is necessary (e.g., a conflict has happened.). The status information for data items can be sent before Package #3 is completely received.

10.3 Refresh Sync from Client Only

The 'refresh sync from client only' is a synchronization type in which the client sends all its data from a database to the server (i.e., exports). The server is expected to replace all data in the target database with the data sent by the client. I.e., this means that the client overwrites all data in the server database.

This refresh sync is treated as a special case of the 'one-way sync from client only'. The only differences between this case and the normal 'one-way sync from client only' are:

1. At the initialization, the sync type (Alert code) MUST be used to indicate that the 'one-way refresh sync from client only' is requested. The Alert code is 203.
2. In Package #3, the Sync element (Pkg #3) from the client to the server MUST include all data from the source database (client database).

10.4 Error Cases Behavior

In this chapter, the recommended behaviors of devices are defined in the cases of different error types.

10.4.1 No Packages from Server after Initialization

See Chapter 9.6.1.

10.4.2 Errors with Defined Error Codes

See Chapter 9.6.4.

11. One-Way Sync from Server only

This sync type is the case in which the client gets all modifications from the server but the client does not send its modifications to the server. Thus, after this type of sync, the client includes all modified data from the server but the server does not know about modifications in the client. In Figure 13, there is depicted the MSC for this scenario.

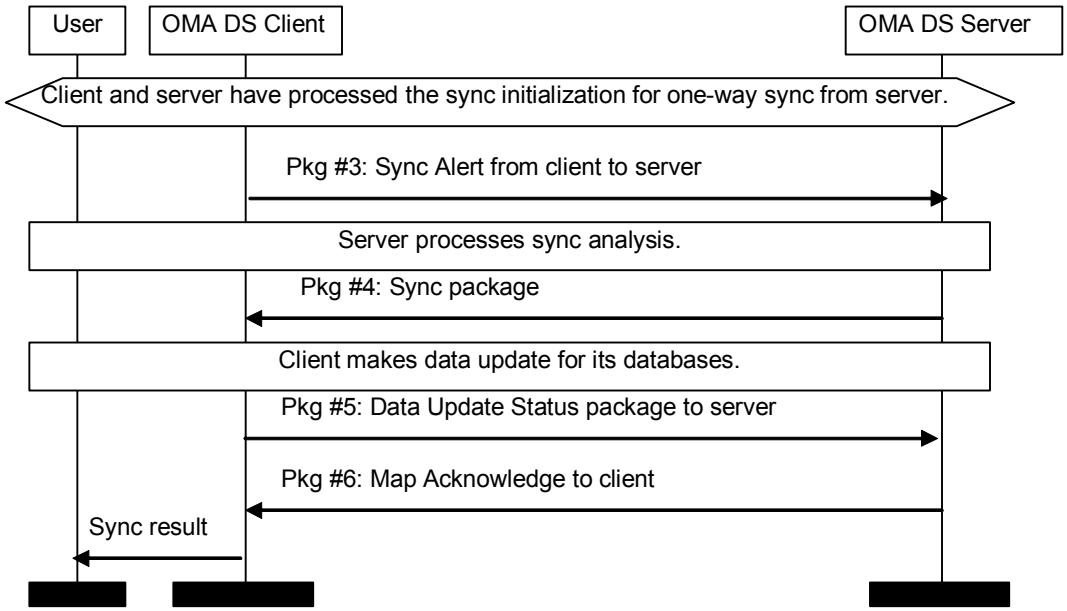

Figure 13 MSC of Sync from Server Only

The package flow presented above is one OMA DS session that means that all messages have the same OMA DS session ID. The Session ID is same as used at the initialization.

The purpose and the requirements for each of package in the figure above are considered in the next sections.

Note. If the sync is done without a separate initialization (See Chapter 6.12), the number of a package in the figure does not describe the actual atomic number of a package in a synchronization session.

11.1 Sync Alert to Server

The sync package (Pkg #3 in Figure 13) is very much similar to the package #3 in the two-way sync but any client modifications are not ever sent to server and the server is only asked to send its modifications to the client. The only difference from the requirements defined in Chapter 9.1 is:

Any client modifications are not included into the Sync element. It MUST be empty.

11.2 Server Modifications to Client

See Chapter 9.2.

11.3 Data Update Status from Client

See Chapter 9.3.

11.4 Map Acknowledge from Server

See Chapter 9.4.

11.5 Refresh Sync from Server Only

The ‘refresh sync from server only’ is a synchronization type in which the server sends all its data from a database to the client. The client is expected to replace all data in the target database with the data sent by the server. This means that the server overwrites all data in the client database.

This refresh sync is treated as a special case of the ‘one-way sync from server only’. The differences between this case and the normal ‘one-way sync from server only’ are:

1. At the Sync Initialization (See Chapter 11.1), the value for the Alert code is 205.
2. In the Server Modifications package to the client (See Chapter 11.2), the Sync element MUST include all data from the source database.
3. The client MUST store all data items to its database (i.e., overwrites old data) and the client MUST return the map items for all stored data items back to the server.

11.6 Error Cases

In this chapter, the recommended behaviors of devices are defined in the cases of different error types.

11.6.1 No Packages from Server

If the client has sent the empty sync command to the server, it does not get any complete response to it (new modifications), the client SHOULD drop the OMA DS session and try to get the modifications later by starting the sync from the beginning.

11.6.2 No Data Update Status from Client

See Chapter 9.6.2.

11.6.3 No Map Ack from Server

See Chapter 9.6.3.

11.6.4 Errors with Defined Error Codes

See Chapter 9.6.4.

12.Server Alerted Sync

This synchronization type is intended to provide the means for a DS Server to alert the DS Client to perform a synchronization. That is, the server informs the client to start synchronization with the server. When the server alerts the client, it also tells it which type of synchronization to initiate. Figure 14 shows the MSC of how Server Alerted Sync is initiated by a server. This is achieved by the server sending a Server Alerted Sync package (also known as Package #0) to the client as described in [SAN].

Figure 14 depicts how a server can initiate a synchronization session.

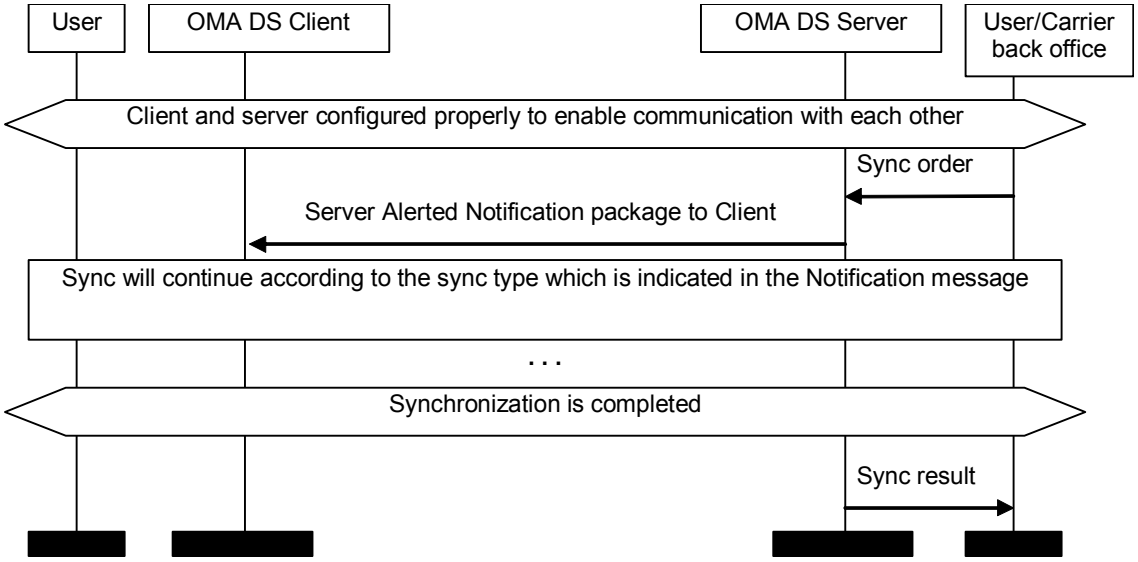

Figure 14 MSC of the Server Alerted Sync session

The package flow presented above is a single OMA DS session. With the exception of the Notification Package itself, all the other packages are the same as they would be for a client-initiated session.

In package #0, a DS Server SHOULD specify a value for the <*sessionid*> field of <*notification-hdr*>, if the server specifies the value, then the client MUST use the same session identifier for all subsequent packages in the session. If a server does not specify a value then it MUST specify a null value (bit value "0000000000000000").

If a server sends more than one Notification Package in an attempt to initiate the same session, then it MUST consistently either send the same value for the <*sessionid*> in each package or specify a null value in each package.

If a client receives more than one package with identical contents then it SHOULD ignore all but one of the packages i.e. it SHOULD only initiate one sync session.

The client MAY respond to the Notification Package.

If a server has sent a Notification Package to a client and does not get a response, the server MAY re-send it.

12.1 Package #0 authentication

Package #0 MUST be authenticated and MD5 Digest authentication MUST be used.

In case the client does not receive the confirmation for the package that delivers the new nonce to the server, the client MUST NOT discard the old nonce as it is unknown on the client side whether the server already received the new nonce or not.

Instead, the client MUST keep both nonces until the new sync notification received from the server and try to authenticate the package #0 using the old nonce if the authentication with the new one fails. If the authentication fails with both nonces, the package #0 is assumed from non-authorized sender and nonces MUST be left intact. If the authentication succeeds, the invalid nonce MUST be discarded on the client and the valid nonce MUST be stored as "old" until the new nonce is created and successfully delivered to the server.

The old nonce MUST be discarded on the client only after the client receives a successful confirmation for the package that contains the new nonce for the server's future use.

12.2 Structure of the Server Alerted Sync Package

Figure 15 describes the format of the Server Alerted Sync Package.

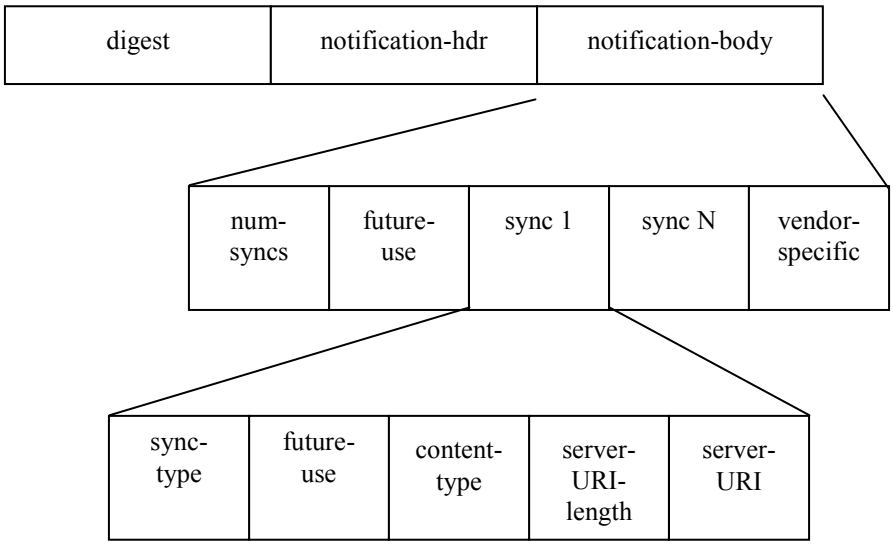

Figure 15. Format of the Server Alerted Sync Package (Pkg #0)

12.3 Syntax for the Package

The following ABNF [RFC2234] defines the syntax for the package. The order and the size of the fields MUST be as specified in the following syntax of the Notification Package.

```
<notification-package> ::= <digest><notification>
<digest> ::= 128*BIT                                    ; 'MD5 Digest value'
<notification> ::= <notification-hdr><notification-body>
<notification-hdr> ::= [specified in [SAN]]
<notification-body> ::= <num-syncs><future-use>*<sync>
                        <vendor-specific>   ; 'Body Data'
<num-syncs> ::= 4*BIT                                   ; 'Number of syncs'
<future-use> ::= 4*BIT                                  ; 'Reserved for future use'
<sync> ::= <sync-type><future-use><content-type>
                    <server-URI-length><server-URI>     ; 'Sync Information'
```

*<sync-type> ::= 4*BIT* ; 'Synchronization type'
<future-use> ::= *4*BIT* ; 'Reserved for future use'
*<content-type> ::= 24*BIT* ; 'Content type'
*<server-URI-length> ::= 8*BIT* ; 'Server URI Length'
*<server-URI> ::= n*BIT* ; 'Server URI
*<vendor-specific> ::= n*BIT* ; 'Optional vendor-specific information'

12.4 Description of the fields

12.4.1 Version

The <*version*> field of the <*notification-hdr*> [[SAN]] MUST specify the version of the OMA Data Synchronization protocol that the server supports.

12.4.2 Notification Package

The <*notification-package*> field specifies the content of the package the server sends to the client indicating the server's intent for the client to start a data synchronization session. Further details can be found in [[SAN]].

12.4.3 Digest

The <*digest*> field specifies the MD5 Digest authentication as described in [RFC1321]. The *server-identifier* element in the digest is the same as the <*server-identifier*> field in the <*nofification-hdr*> [SAN].

Further details of the use of the digest and authentication can be found in [REPPRO].

12.4.4 Notification

The <*notification*> field is the container for the <*notification-hdr*> and <*notification-body*>.

12.4.5 Header of the Notification Package

The <*notification-hdr*> field specifies the header of the Notification Package as described in [SAN].

The value for the Server-Id field MUST be unique. It MUST be the same as the Server Identifier (DevID) of the DevInf DEVINF.

It SHOULD contain the server domain name to aid uniqueness.

It SHOULD be kept as short as possible for efficiency and cannot exceed 255 characters in length.

12.4.6 Body of the Notification Package

The <*notification-body*> field specifies the body of the Notification Package.

12.4.7 Number of syncs

The <*num-syncs*> field is used to specify the number of synchronizations that the server requests the client to initiate. In other words the number of data stores that are to be synchronized.

If the server is requesting that N data stores to be synchronized then the <*notification-body*> MUST contain N <*sync*> fields.

If the server is requesting that all of the client's data stores to be synchronized then <*num-syncs*> field MUST be set to zero (bit value "0000") and the notification package MUST NOT contain any <*sync*> fields i.e. the contents of the <*notification-body*> field is <*num-syncs*><*future-use*><*vendor-specific*>.

If <*num-syncs*> is zero then

- the client SHOULD use a 'two-way' sync type (value 200), unless the client can determine a more appropriate action.
- the client SHOULD initiate synchronization with all known data stores of the server. Data stores could be known as a result of an earlier sync session, or as configured by an end-user.

12.4.8 Future Use

The <*future-use*> field is reserved for future use for OMA Data Synchronization. The reserved space is 4 bits long and the value for bits not yet in use MUST be "0".

12.4.9 Vendor specific information

The optional <*vendor specific*> field is used to specify vendor-specific information.

12.4.10 Sync Info

The <*sync*> field is a container for the <*sync-type*>, <*future-use*>, <*content-type*>, <*server-URI-length*> and <*server-URI*> fields. The number of sync fields in the Notification Package MUST be equal to the value specified in the <*num-syncs*> field.

12.4.11 Sync Type

The <*sync-type*> field specifies the synchronization type the server is requesting the client to start. The value used in this field MUST be the numeric value computed using the corresponding alert code 206-210 (see Section 13.2) minus 200. For example, for a server to indicate a 'two-way sync by server' (alert code is 206) the <*sync-type*> field would contain the value 6.

When the device starts a synchronization session in response to receipt of a Notification Package then in its initialization package sent to the server it MUST use the value of alert code used at the synchronization initialization (see Section 13.2) and MUST NOT use the value of <*sync type*> in the notification package. For example, if the value of <*sync-type*> is 6 ('two-way sync by server'), then the initialization package would contain the alert code for the 'two-way sync' (value 200).

The client MAY initiate a synchronization session with an alert code which does not correspond to <*sync type*> value. For example, if the value of <*sync type*> is 'two-way sync by server' (value 6) then the initialization package may contain the alert code for 'one-way from client' (value 202).

12.4.12 Future Use

The <*future-use*> field is reserved for future use for OMA Data Synchronization. The reserved space is 4 bits long and the value for bits not yet in use MUST be "0".

12.4.13 Content Type

The <*content-type*> field specifies the MIME media content type of data object that the server instructs the device to synchronize.

The <content-type> field SHOULD be used to indicate the content type of the data store the server wishes to synchronize. If not specified then it MUST contain zero (bit value "000000000000000000000000"). If the content type is being specified

then it MUST contain the numeric content type code (described in [WSPCTC]) and not its corresponding textural representation.

12.4.14 Server URI length

The <*server-URI-length*> field specifies the length of the name of the server data store in bytes. It MUST be specified for each <*sync*> item present in the <*notification-body*> field.

12.4.15 Server URI

The <*server-URI*> field specifies the name of the server data store. Relative URI's SHOULD be used for efficiency. The length of this field is specified in the <*server-URI-length*> field, and cannot exceed 255 characters. It MUST be specified for each <*sync*> item present in the <*notification-body*> field.

12.5 Example of Server Alerted Sync Notification Package

<< Data (i.e. notification) pushed from the server >>		
XX, XX, XX, XX, XX, XX, XX, XX, XX, XX, XX, XX, XX, XX, XX, XX	128-bit digest value	MD5 Digest authentication value
Notification Header >>		
03, 18, 00, 00, 00	Binary '0000001100'	Version '1.2'
	Binary '01' = Background mode	UI-Mode '1'
	Binary '1' = Server Initiated Action	Initiator '1'
	Binary '00000000000000000000000000'	Future use
00, 01	Binary '0000000000000001'	SessionID '1'
0F	Binary '00001111'	Server Identifier length '15'
73, 79, 6E, 63, 2E, 73, 65, 72, 76, 65, 72, 2E, 63, 6F, 6D	String 'sync.server.com'	Server Identifier
Notification Body >>		
10	Binary '0001'	Number of syncs. One Sync Info container included
	Binary '0000'	Future use
Sync info >>		
60	Binary '0110'	Sync type '206'
	Binary '0000'	Future use
00, 00, 07	Binary '000000000000000000000111'	Content-type 'text/x-vcard'

04	Binary ‘00000100’	Server URI length ‘4’
2F, 63, 6F, 6E	String ‘/con’	The name of server data store [Server URI]
*Optional Vendor specific info >> (Binary n*BIT)*		

13.Protocol Values and Alert Codes

13.1 Protocol Values

Here are listed all protocol values (string values), which can be used in the VerProto element. The protocol version 1.2 is used by the implementations complying with this specification

VerProto Codes	Description
SyncML/1.0	Indicates that this SyncML message uses the sync protocol 1.0 defined by the SyncML Initiative.
SyncML/1.1	Indicates that this SyncML message uses the sync protocol 1.1 defined by the SyncML Initiative.
SyncML/1.2	Indicates that this SyncML message uses the sync protocol 1.2 defined by the OMA DS WG.

13.2 Alert Codes

Here are listed all Alert codes and values, which are used in the Data element when the Alert command is sent.

Alert Code Value	Name	Description
Alert Codes used for user alerts		
100	DISPLAY	Show. The Data element type contains content information that should be processed and displayed through the user agent.
101-150	-	Reserved for future SyncML usage.
Alert Codes used at the synchronization initialization		
200	TWO-WAY	Specifies a client-initiated, two-way sync.
201	SLOW SYNC	Specifies a client-initiated, two-way slow-sync.
202	ONE-WAY FROM CLIENT	Specifies the client-initiated, one-way only sync from the client to the server.
203	REFRESH FROM CLIENT	Specifies the client-initiated, refresh operation for the one-way only sync from the client to the server.
204	ONE-WAY FROM SERVER	Specifies the client-initiated, one-way only sync from the server to the client.
205	REFRESH FROM SERVER	Specifies the client-initiated, refresh operation of the one-way only sync from the server to the client.
Alert Codes used by the server when alerting the sync.		
206	TWO-WAY BY SERVER	Specifies a server-initiated, two-way sync.
207	ONE-WAY FROM CLIENT BY SERVER	Specifies the server-initiated, one-way only sync from the client to the server.
208	REFRESH FROM CLIENT BY SERVER	Specifies the server-initiated, refresh operation for the one-way only sync from the client to the server.

209	ONE-WAY FROM SERVER BY SERVER	Specifies the server-initiated, one-way only sync from the server to the client.
210	REFRESH FROM SERVER BY SERVER	Specifies the server-initiated, refresh operation of the one-way only sync from the server to the client.
211-220	-	Reserved for future SyncML usage.
Special Alert Codes		
221	RESULT ALERT	Specifies a request for sync results.
222	NEXT MESSAGE	Specifies a request for the next message in the package.
223	NO END OF DATA	End of Data for chunked object not received.
224	SUSPEND	Suspend Synchronization session
225	RESUME	Resume Synchronization session
226-250	-	Reserved for future SyncML usage.

14.Examples

14.1 WBXML Example

Here is an example of Package #3 in tokenized form (numbers in hexadecimal). This example uses opaque data and inline strings. The example also assumes that the character encoding is UTF-8.

```
02 00 00 6A 1E "-" "/" "/" "S" "Y" "N" "C" "M" "L" "/" "/" "D" "T" "D" " " "S" "y" "n" "c"
"M" "L" " " "1" "." "2" "/" "/" "E" "N" 00 6D 6C 71 C3 03 "1" "." "2" 01 72 C3 0A "S" "y" "n"
"c" "M" "L" "/" "1" "." "2" 01 65 C3 01 "1" 01 5B C3 01 "2" 01 6E 57 C3 20 "h" "t" "t" "p"
":" "/" "/" "w" "w" "w" "1" "." "d" "a" "t" "a" "s" "y" "n" "c" "." "o" "r" "g" "/" "s" "e"
"r" "v" "l" "e" "t" 01 01 67 57 C3 12 "I" "M" "E" "I" ":" "1" "5" "6" "4" "4" "6" "9" "2" "1"
"0" "9" "4" "8" 01 01 01 6B 69 4B C3 01 "1" 01 5C C3 01 "1" 01 4C C3 01 "0" 01 4A C3 07 "S"
"y" "n" "c" "H" "d" "r" 01 6F C3 12 "I" "M" "E" "I" ":" "1" "5" "6" "4" "4" "6" "9" "2" "1"
"0" "9" "4" "8" 01 68 C3 20 "h" "t" "t" "p" ":" "/" "/" "w" "w" "w" "1" "." "d" "a" "t" "a"
"s" "y" "n" "c" "." "o" "r" "g" "/" "s" "e" "r" "v" "l" "e" "t" 01 4F C3 3 "2" "0" "0" 01 01
69 4B C3 01 "2" 01 5C C3 01 "1" 01 4C C3 01 "1" 01 4A C3 05 "A" "l" "e" "r" "t" 01 6F C3 0E
"." "\" "d" "e" "v" "-" "c" "a" "l" "e" "n" "d" "a" "r" 01 68 C3 0A "." "/" "c" "a" "l" "e"
"n" "d" "a" "r" 01 4F C3 03 "2" "0" "0" 01 54 4F 00 02 4A C3 11 "2" "0" "0" "0" "0" "5" "0"
"2" "2" "T" "0" "9" "3" "2" "2" "3" "Z" 01 00 00 01 01 01 6A 4B C3 01 "3" 01 6E 57 C3 0A "."
"/" "c" "a" "l" "e" "n" "d" "a" "r" 01 01 67 57 C3 0E "." "\" "d" "e" "v" "-" "c" "a" "l" "e"
"n" "d" "a" "r" 01 01 33 C3 01 "1" 01 60 4B C3 01 "4" 01 5A 00 02 4D 03 "t" "e" "x" "t" "/"
"x" "-" "v" "c" "a" "l" "e" "n" "d" "a" "r" 00 01 00 00 01 54 67 57 C3 02 "2" "6" 01 01 4F C3
02 04 "C" "A" "L" "1" 01 01 01 01 12 01 01 01
```

In an expanded and annotated form:

Token Stream	Description
02	Version number – WBXML v1.2
00	FPI for DTD in string table
00	index into string table for the identifier
6A	Charset is UTF-8
1E	String table length
"-" "/" "/" "S" "Y" "N" "C" "M" "L" "/" "/" "D" "T" "D" " " "S" "y" "n" "c" "M" "L" " " "1" "." "2" "/" "/" "E" "N" 0x00	-//SYNCML//DTD SyncML 1.2//EN
6D	<SyncML>
6C	<SyncHdr>
71	<VerDTD>
C3	Opaque data follows
03	Length of opaque data
"1" "." "2"	String '1.2'
01	</VerDTD>
72	<VerProto>
C3	Opaque data follows
0A	Length of opaque data
"S" "y" "n" "c" "M" "L" "/" "1" "." "2"	String 'SyncML/1.2'
01	</VerProto>
65	<SessionID>
C3	Opaque data follows
01	Length of opaque data
"1"	String '1'
01	</SessionID>
5B	<MsgID>
C3	Opaque data follows
01	Length of opaque data
"2"	String '2'
01	</MsgID>
6E	<Target>
57	<LocURI>

C3	Opaque data follows
20	Length of opaque data
"h" "t" "t" "p" ":" "/" "/" "w" "w" "w" "1" "." "d" "a" "t" "a" "s" "y" "n" "c" "." "o" "r" "g" "/" "s" "e" "r" "v" "l" "e" "t"	String 'http://www1.datasync.org/servlet"
01	</LocURI>
01	</Target>
67	<Source>
57	<LocURI>
C3	Opaque data follows
12	Length of opaque data
"I" "M" "E" "I" ":" "1" "5" "6" "4" "4" "6" "9" "2" "1" "0" "9" "4" "8"	String 'IMEI:1564469210948'
01	</LocURI>
01	</Source>
01	</SyncHdr>
6B	<SyncBody>
69	<Status>
4B	<CmdID>
C3	Opaque data follows
01	Length of opaque data
"1"	String '1'
01	</CmdID>
5C	<MsgRef>
C3	Opaque data follows
01	Length of opaque data
"1"	String '1'
01	</MsgRef>
4C	<CmdRef>
C3	Opaque data follows
01	Length of opaque data
"0"	String '0'
01	</CmdRef>
4A	<Cmd>
C3	Opaque data follows
07	Length of opaque data
"S" "y" "n" "c" "H" "d" "r"	String 'SyncHdr'
01	</Cmd>
6F	<TargetRef>
C3	Opaque data follows
12	Length of opaque data
"I" "M" "E" "I" ":" "1" "5" "6" "4" "4" "6" "9" "2" "1" "0" "9" "4" "8"	String 'IMEI:1564469210948'
01	</TargetRef>
68	<SourceRef>
C3	Opaque data follows
20	Length of opaque data
"h" "t" "t" "p" ":" "/" "/" "w" "w" "w" "1" "." "d" "a" "t" "a" "s" "y" "n" "c" "." "o" "r" "g" "/" "s" "e" "r" "v" "l" "e" "t"	String 'http://www1.datasync.org/servlet"
01	</LocURI>
4F	<Data>
C3	Opaque data follows
3	Length of opaque data
"2" "0" "0"	String '200'
01	</Data>
01	</Status>
69	<Status>
4B	<CmdID>
C3	Opaque data follows
01	Length of opaque data
"2"	String '2'

01	</CmdID>
5C	<MsgRef>
C3	Opaque data follows
01	Length of opaque data
"1"	String '1'
01	</MsgRef>
4C	<CmdRef>
C3	Opaque data follows
01	Length of opaque data
"1"	String '0'
01	</CmdRef>
4A	<Cmd>
C3	Opaque data follows
05	Length of opaque data
"A" "l" "e" "r" "t"	String 'Alert'
01	</Cmd>
6F	<TargetRef>
C3	Opaque data follows
0E	Length of opaque data
"." "\" "d" "e" "v" "-" "c" "a" "l" "e" "n" "d" "a" "r"	String '.\dev-calendar'
01	</TargetRef>
68	<SourceRef>
C3	Opaque data follows
0A	Length of opaque data
"." "/" "c" "a" "l" "e" "n" "d" "a" "r"	String './calendar'
01	</LocURI>
4F	<Data>
C3	Opaque data follows
03	Length of opaque data
"2" "0" "0"	String '200'
01	</Data>
54	<Item>
4F	<Data>
00	Switch codepage
01	Codepage 01 (MetInf)
4F	<Next>
C3	Opaque data follows
11	Length of opaque data
"2" "0" "0" "0" "0" "5" "0" "2" "2" "T" "0" "9" "3" "2" "2" "3" "Z"	String '200005022T093223Z '
01	</Next>
00	Switch codepage
00	Codepage 00
01	</Data>
01	</Item>
01	</Status>
6A	<Sync>
4B	<CmdID>
C3	Opaque data follows
01	Length of opaque data
"3"	String '3'
01	</CmdID>
6E	<Target>
57	<LocURI>
C3	Opaque data follows
0A	Length of opaque data
"." "/" "c" "a" "l" "e" "n" "d" "a" "r"	String './calendar'
01	</LocURI>
01	</Target>
67	<Source>
57	<LocURI>

C3	Opaque data follows
0E	Length of opaque data
"." "\" "d" "e" "v" "-" "c" "a" "l" "e" "n" "d" "a" "r"	String '.\dev-calendar'
01	</LocURI>
01	</Source>
33	<NumberOfChanges>
C3	Opaque data follows
01	length of opaque data
"1"	String '1'
01	</NumberOfChanges>
60	<Replace>
4B	<CmdID>
C3	Opaque data follows
01	Length of opaque data
"4"	String '4'
01	</CmdID>
5A	<Meta>
00	Codepage switch
01	Codepage 01 (MetInf)
4D	<Type>
03	Inline string follows
"t" "e" "x" "t" "/" "x" "-" "v" "c" "a" "l" "e" "n" "d" "a" "r" 00	String 'text/x-vcalendar'
01	</Type>
00	Codepage switch
00	Codepage 00
01	</Meta>
54	<Item>
67	<Source>
57	<LocURI>
C3	Opaque data follows
02	Length of opaque data
"2" "6"	String '26'
01	</LocURI>
01	</Source>
4F	<Data>
C3	Opaque data follows
02	Length of opaque data
04	Legnth of string table
"C" "A" "L" "1"	Actual data
01	</Data>
01	</Item>
01	</Replace>
01	</Sync>
12	<Final>
01	</Final>
01	</SyncBody>
01	</SyncML>

Appendix A. Static Conformance Requirements (Normative)

A.1 Conformance Requirements for OMA DS Client

Table 1 – Client Features

Item	Functionality	Reference	Status	Requirement
SCR-DS-CLIENT-001	Support of 'two-way sync' sync type	8	M	
SCR-DS-CLIENT-002	Support of 'slow two-way sync' sync type	9.5	M	
SCR-DS-CLIENT-003	Support of 'one-way sync from client only' sync type	10	O	
SCR-DS-CLIENT-004	Support of 'refresh sync from client only' sync type	10.3	O	
SCR-DS-CLIENT-005	Support of 'one-way sync from server only' sync type	11	O	
SCR-DS-CLIENT-006	Support of 'refresh sync from server only' sync type	11.5	O	
SCR-DS-CLIENT-007	Support of 'sync alert' For non-WAP clients, or WAP clients that only use OBEX transport for DS.	12	O	
SCR-DS-CLIENT-008	Support of 'Sync Without Separate Initialization'	6.11	O	
SCR-DS-CLIENT-009	Support of sending 'Large Objects'	6.10	O	SCR-DS-CLIENT-LO-S-004 SCR-DS-CLIENT-LO-S-005 SCR-DS-CLIENT-LO-S-006 SCR-DS-CLIENT-LO-S-007
SCR-DS-CLIENT-010	Support of receiving 'Large Objects'	6.10	O	SCR-DS-CLIENT-LO-R-001 SCR-DS-CLIENT-LO-R-002 SCR-DS-CLIENT-LO-R-003
SCR-DS-CLIENT-011	Support of 'busy signaling'	6.13	O	
SCR-DS-CLIENT-012	Support of suspend/resume	6.12	O	
SCR-DS-CLIENT-013	Support for filtering		O	SCR-DS-COMMON-C-008
SCR-DS-CLIENT-014	Support for hierarchical synchronization		O	SCR-DS-CONTENT-C-007
SCR-DS-CLIENT-015	Support WAP PUSH operation For WAP capable devices that use HTTP or WSP transport for DS	12	M	DSDM-WSP-C-002

SCR-DS-CLIENT-016	Support of 'sync alert' by WAP Push method For WAP capable devices that use HTTP or WSP transport for DS	12	M	DSDM-WSP-C-002

A.1.1 SCR for Large Object

Item	Function	Reference	Status	Requirement
SCR-DS-CLIENT-LO-R-001	Indicate support for receiving Large Object in the DevInf	6.10	M	SCR-DS-DEVINF-C-029
SCR-DS-CLIENT-LO-R-002	Sending MaxObjSize and MaxMsgSize	6.10	M	
SCR-DS-CLIENT-LO-R-003	Sync Commands inside Large Object is handled as Atomic	6.10	M	
SCR-DS-CLIENT-LO-S-004	Data chunks must be sent in continuous order without any new command	6.10	M	
SCR-DS-CLIENT-LO-S-005	Include Size in the first data chunk	6.10	M	
SCR-DS-CLIENT-LO-S-006	All chunks except the last one must include "MoreData" tag	6.10	M	
SCR-DS-CLIENT-LO-S-007	Repeat Meta and Item information in each chunk	6.10	O	

A.2 Conformance Requirements for OMA DS Server

Table 2 – Server Features

Item	Functionality	Reference	Status	Requirement
SCR-DS-SERVER-001	Support of 'two-way sync' sync type	8	M	
SCR-DS-SERVER-002	Support of 'slow two-way sync' sync type	9.5	M	
SCR-DS-SERVER-003	Support of 'one-way sync from client only' sync type	10	M	
SCR-DS-SERVER-004	Support of 'refresh sync from client only' sync type	10.3	M	
SCR-DS-SERVER-005	Support of 'one-way sync from server only' sync type	11	M	
SCR-DS-SERVER-006	Support of 'refresh sync from server only' sync type	11.5	M	
SCR-DS-SERVER-007	Support of 'sync alert'	12	O	

SCR-DS-SERVER-008	Support of 'Sync Without Separate Initialization'	6.11	M	
SCR-DS-SERVER-009	Support of sending 'Large Objects'	6.10	O	SCR-DS-SERVER -LO-S-004 SCR-DS-SERVER -LO-S-005 SCR-DS-SERVER -LO-S-006 SCR-DS-SERVER -LO-S-007
SCR-DS-SERVER-010	Support receiving Large Object Handling	6.10	M	- SCR-DS-SERVER -LO-R-001 SCR-DS-SERVER -LO-R-002 SCR-DS-SERVER -LO-R-003
SCR-DS-SERVER-011	Support of 'busy signaling'	6.13	O	
SCR-DS-SERVER-012	Support of suspend/resume	6.12	M	
SCR-DS-SERVER-013	Support for filtering		O	SCR-DS-COMMON-S-008
SCR-DS-SERVER-014	Support for hierarchical synchronization		O	SCR-DS-CONTENT-S-007

A.2.1 SCR for Large Object

Item	Function	Reference	Status	Requirement
SCR-DS-SERVER-LO-R-001	Indicate support for Large Object in DevInf	6.10	M	SCR-DS-DEVINF-S-029
SCR-DS-SERVER-LO-R-002	Sending MaxObjSize and MaxMsgSize	6.10	M	
SCR-DS-SERVER-LO-R-003	Sync Command inside Large Object is handled as Atomic	6.10	M	
SCR-DS-SERVER-LO-S-004	Data chunks must be sent in continuous order without any new command	6.10	M	
SCR-DS-SERVER-LO-S-005	Include Size in the first data chunk	6.10	M	
SCR-DS-SERVER-LO-S-006	All chunks except the last one must include "MoreData"	6.10	M	
SCR-DS-SERVER-LO-S-007	Repeat Meta and Item information in each chunk	6.10	O	

Appendix B. Change History (Informative)

B.1 Approved Version History

Reference	Date	Description
OMA-SyncML-DataSyncProtocol–V1_1_2-20030612-A	12 Jun 2003	Initial OMA release
OMA-TS-DS_Protocol-V1_2-20060710-A	10 Jul 2006	Approved by TP ref#OMA-TP-2006-0239R03-INP_DS_V1_2_for_final_approval
OMA-TS-DS_Protocol-V1_2_1-20070614-A	14 June 2007	Icorporated CRs: OMA-DS-DS_1_2-2006-0018R01 OMA-DS-DS_1_2-2006-0019 OMA-DS-DS_1_2-2006-0033R02 OMA-DS-DS_1_2-2007-0001R01 OMA-DS-DS_1_2-2007-0009 OMA-DS-DS_1_2-2007-0012R01 OMA-DS-DS_1_2-2007-0017 OMA-DS-DS_1_2-2007-0018R01 OMA-DS-DS_1_2-2007-0019R02 OMA-DS-DS_1_2-2007-0023 OMA-DS-DS_1_2-2007-0030
OMA-TS-DS_Protocol-V1_2_1-20070810-A	10 Aug 2007	Editorial change correcting figure numbering. Prepared for TP notification TP ref # OMA-TP-2007-0326-INP_DS_V1_2_1_ERP_for_Notification