SyncML Sync Protocol, version 1.1

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

SyncML Initiative

The following companies are Sponsors of the SyncML Initiative:

Ericsson
IBM
Lotus
Matsushita Communications Industrial Co., Ltd.
Motorola
Nokia
Openwave
Starfish Software
Symbian

Revision History

Revision	Date	Comments
0.9	2000-05-31	0.9 release
1.0a	2000-08-31	Authentication procedures added, busy signaling generalized, multiple message per package functionality specified, Update command renamed to Replace, Alert codes modified, editorial changes.
1.0b	2000-11-07	Sync Anchors chapter updated, error cases fixed, slow sync chapter fixed, the sync alert chapter updated, examples updated
1.0	2000-12-07	The candidate version for the final release. The authentication example fixed. The device capabilities and the requirement for Get operation changed. Binary example updated. Examples updated to match with changes in the DevInf and MetInf specifications
1.0.1	2001-05-28	Incorporated Erratas
1.0.1	2001-05-03	Fixed copyright dates.
1.1	2002-02-15	The candidate version for the 1.1 release. Incorporated errata, changed the value of VerDTD and VerProto to 1.1. Fixed copyright text, device info URI to point to devinf11, versions in the wbxml example. Corrected the MD5 example. Added in a paragraph defining Large Object delivery, fixed the WBXML example. Modified server alerted sync.

Copyright Notice

Table of Contents

1 Introduction

The purpose of this specification is to define a synchronization protocol using the SyncML Representation protocol [1]. This protocol is called the SyncML Sync Protocol. This specification defines the protocol for different sync procedures, which can occur between a SyncML client and a SyncML server, in the form of message sequence charts (MSC's). The specification covers the most useful and common synchronization cases (Chapters 4-8).

1.1 SyncML Framework

This specification can be implemented by using the SyncML interface from the SyncML Framework (See Figure 1). Not all the features of the SyncML Interface are required to comply with this specification.

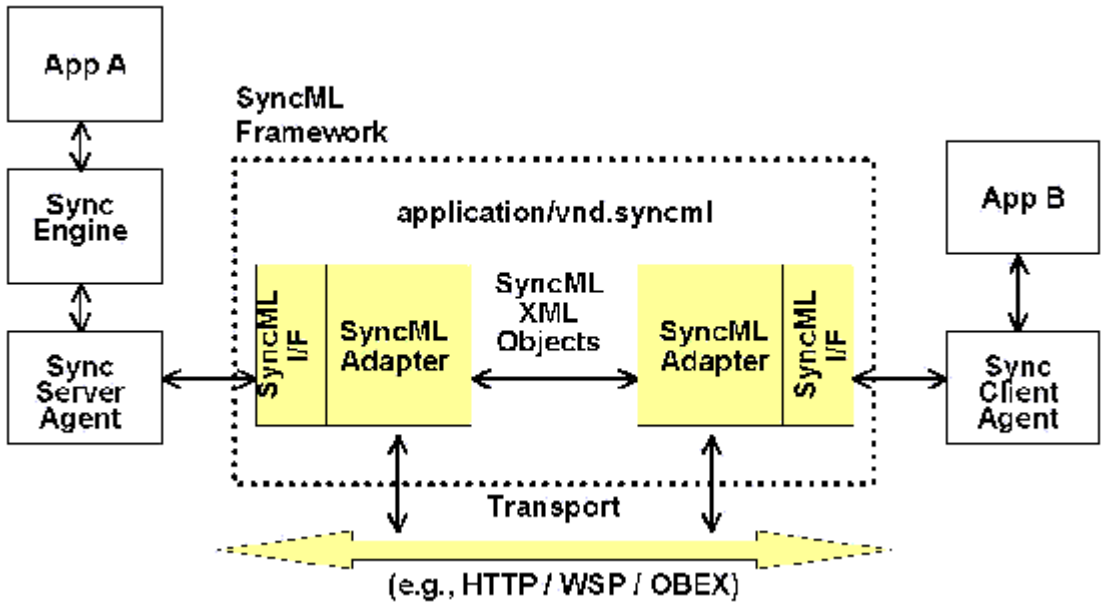

Figure 1 SyncML Framework

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 SyncML server side even if some SyncML client implementations may 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 [1] offered by the SyncML interface ('SyncML I/F') [2] to communicate with each other.

1.2 Device Roles

Figure 2 depicts a synchronization example in which a mobile phone acts as a SyncML client and a server acts as a SyncML server. The SyncML client sends SyncML message including the data modifications made in the client to the SyncML 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 SyncML server returns its modifications back to the SyncML client.

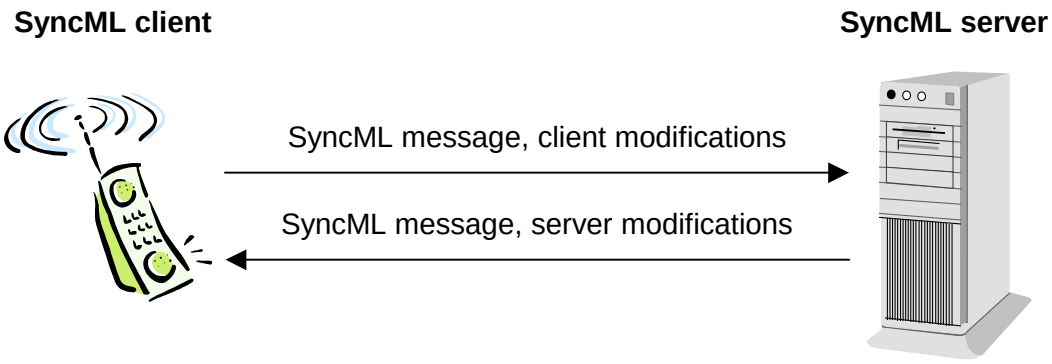

Figure 2 Synchronization Example with Mobile Phone and Server

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

SyncML 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 SyncML server. Although the SyncML client has always the role to send its modifications first, in some cases the server may have a role to initiate synchronization. The SyncML client is typically a mobile phone, PC, or PDA device.

SyncML Server – This is the device, which contains a sync server agent and sync engine, and which usually waits that the SyncML 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 may be able to initiate synchronization if unsolicited commands from the server to the client are supported on the transport protocol level. Typically, the SyncML server is a server device or PC.

1.3 Sync Types

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

Table 1 SyncML 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 5
Slow sync	A form of two-way sync in which all items are compared with each other on a field-by-field basis. In practise, 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 5.5
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 6

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 6.3
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 7
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 7.5
Server Alerted Sync	A sync type in which the server to alerts the client to perform sync. That is, the server informs the client to starts a specific type of sync with the server.	Chapter 8

1.4 Symbols and conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY" and "OPTIONAL" in this document are to be interpreted as described in [RFC 2119].

Any reference to components of the Device Information DTD or XML snippets are specified in this `type face`.

1.4.1 MSC Notation

Notation used in the message sequence charts:

Box – Indicates the start of a procedure or an internal process in a device

Hexagon – Indicates a condition that is needed to start the transaction below this hexagon

Arrow – Represents a message, or transaction

2 Protocol Fundamentals

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

2.1 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 2.3). To indicate the type of a modification, the different operations (e.g., Replace) are used.

2.1.1 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.

2.2 Usage of Sync Anchors

2.2.1 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 Information DTD [3]), 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 Initiative. 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 SyncML 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.

2.2.1.1 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 #2, 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.

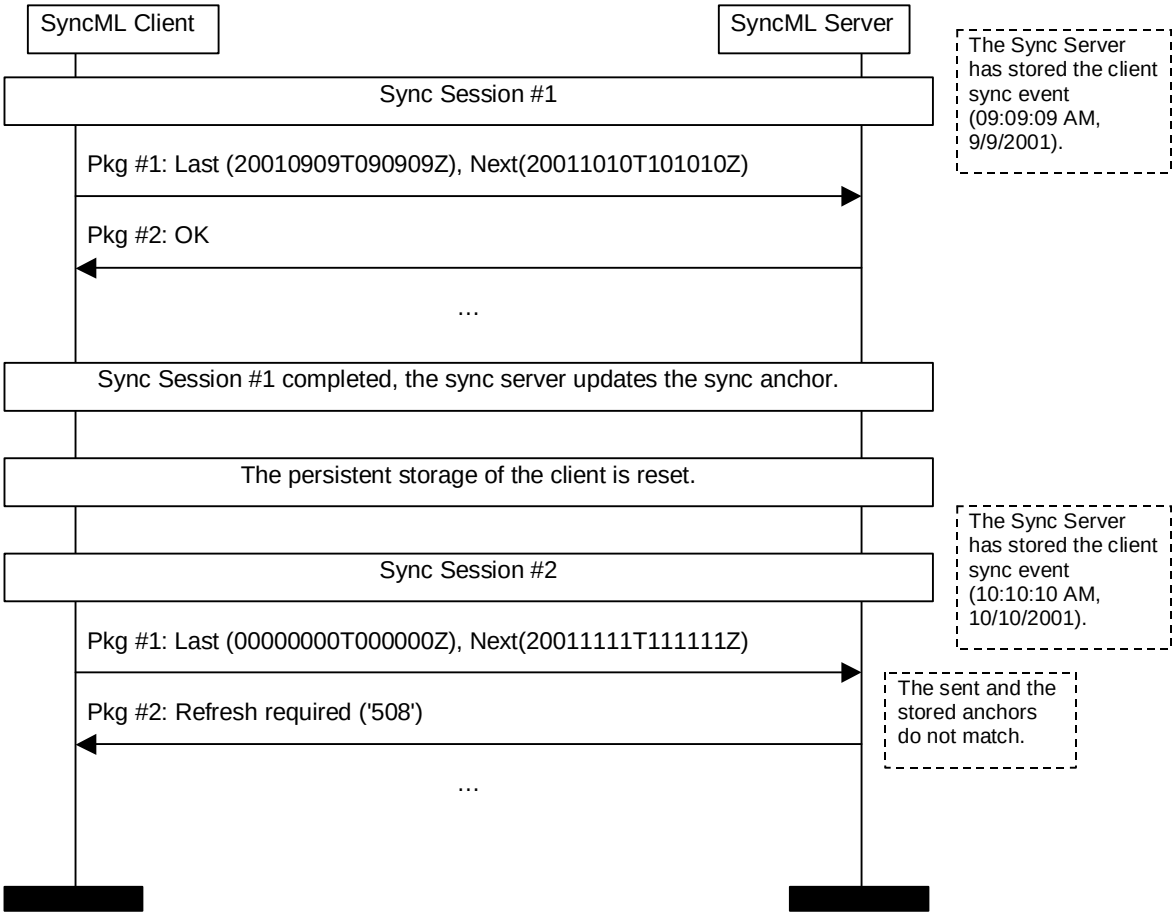

Figure 3 Example of Sync Anchor Usage

2.2.2 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 [5].

2.3 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 4 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.

Client Device

Client Database:

LUID	Data
11	Car
22	Bike
33	Truck
44	Shoes

Server Device

Server Database:

GUID	Data
1010101	Car
2121212	Bike
3232323	Truck
4343434	Shoes

Server Mapping Table:

GUID	LUID
1010101	11
2121212	22
3232323	33
4343434	44

Figure 4 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.

2.3.1 Caching of Map Operations

After a SyncML server has requested one or more additions to be done by the SyncML 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 SyncML 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.

2.4 Conflict Resolution

Conflicts, which happen because of modifications on the same items on the server and the client databases, are in general resolved by a sync engine SW on the server device. This protocol with the SyncML Representation protocol provides the functionality to notify the SyncML client about the resolved conflicts.

Although the SyncML 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 13 in [1]) 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.

2.5 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 can challenge for the authentication and the device receiving the authentication challenge must be able to send the authorization credentials back.

The authentication procedure used by this protocol is defined in Chapter 3.

2.6 Addressing

2.6.1 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 an 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) does not match with the device or server address, if this type of scheme is used.

2.6.1.1 Usage of RespURI and Re-direction Status Codes

This protocol requires that the devices support receiving the RespURI element as specified in the SyncML Representation Protocol specification, but the support of the re-direction status codes (3XX) is not required.

2.6.2 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>
```

2.6.3 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>
```

2.7 Exchange of Device Capabilities

This protocol provides the functionality exchange the device capabilities during the initialization (See Chapter 4). The exchange can be requested by the sync client or the sync server.

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 it is asked by the server. The client SHOULD also support the receiving of the server device information.

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 when sent by the client or requested by the server itself.

Implementation consideration. The exchange of the device information can require that a quite large amount of data is transferred over the air. Thus, the devices should avoid requesting the exchange at every time when 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.

2.8 Device Memory Management

This protocol with the Meta Information DTD provides 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 when the synchronization is done. The static memory capabilities are exchanged when the sync initialization is done (See Chapter 2.7 and Chapter 4).

Although the sending of persistent memory capabilities is optional for both the sync clients and servers, the sync clients SHOULD send those and the sync servers MAY.

The usage of different types of memory capabilities is dependent on the persistent storage model on a device. Below there is an example 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. Thus, the database is specified inside the Meta element anymore.

2.9 Multiple Messages in Package

This protocol provides the functionality to transfer one SyncML package in multiple SyncML messages. This may be necessary if one SyncML package is too large to be transferred in one SyncML message. This limitation may 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 SyncML Representation protocol.). Other messages belonging to the package MUST NOT include the Final element. The Final element must 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 must not 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 may not be desired 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 3-7, 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).

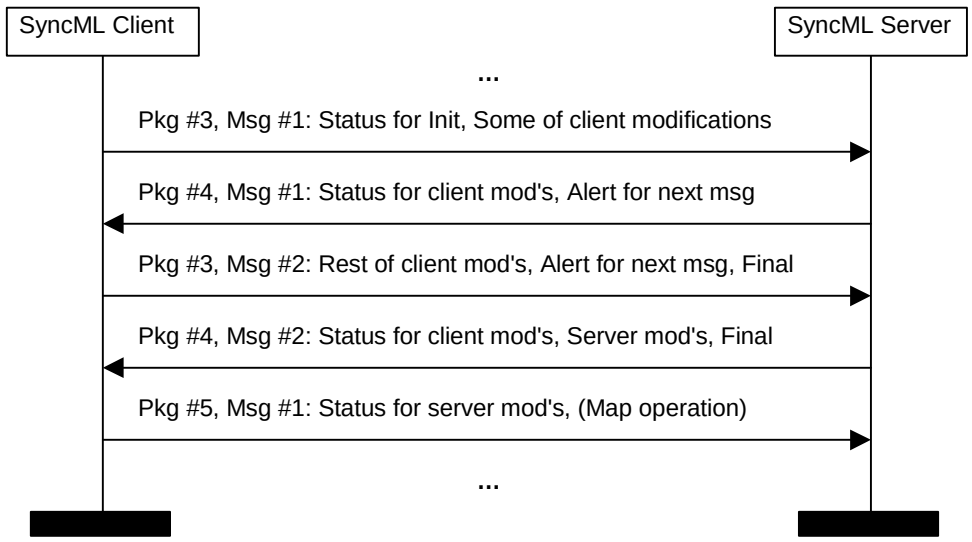

Figure 5 Example of Sending Multiple Messages in a Package

2.10 Large Object Handling

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

On receipt of a data object with the <MoreData/> element, the recipient MUST respond with a status response “214 – Chunked item accepted and buffered” and, if there are no other commands to be sent, ask for the next message using the Alert 222 mechanism defined in section 2.9.

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

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

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

Meta and Item information SHOULD be repeated on each subsequent message containing chunks of the same data object. Authentication details related to the data object may vary between messages bearing chunks of the same data object as defined in the section 3.

If a device supports Large Object Handling, it MUST declare the maximum size of object it is capable of receiving as Meta information within the Alert or Sync command, as specified in [3].

Servers and clients SHOULD use knowledge of their own and the recipient's MaxMsgSize to determine at what size segmentation must occur. If an item is chunked across multiple messages, the <Size> element of the Meta information MUST be used to signal to the recipient the overall size of the data object. The <Size> element MUST only be specified in the first chunk of the item.

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

If the recipient detects a new data object or command before the previous item has been completed (by the chunk without the <MoreData/> Element), the recipient MUST respond with an Alert 223 – "End of Data for chunked object not received". The Alert should contain the source and/or target information from the original command to enable the sender to identify the failed command. Note: a Status would not suffice here because there would not necessarily be a command ID to refer to. The recipient MUST NOT commit the command. The sender MAY attempt to retransmit the entire data object.

Below is an example segment of message exchange in which a large object is added:

```
<Add>
 <CmdID>15</CmdID>
 <Meta>
  <Type>text/x-vcard</Type>
  <size>3000</size>
 </Meta>
 <Item>
  <Source>
   <LocURI>2</LocURI>
  </Source>
  <Data>BEGIN:VCARD
VERSION:2.1
FN:Bruce Smith
N:Smith;Bruce
TEL;WORK;VOICE:+1-919-555-1234
TEL;WORK;FAX:+1-919-555-9876
NOTE: here starts a huge note field, or icon etc...
  </Data>
  <MoreData/>
</Item>
</Add>
```

Recipient would send back its command(s) or an alert 222.

```
...
<Add>
 <CmdID>28</CmdID>
 <Item>
  <Source>
   <LocURI>2</LocURI>
  </Source>
  <Data>here is the rest of the huge field.....
blah, blah, blah.....
END:VCARD
  </Data>
 </Item>
</Add>
...
```

2.11 Sync without Separate Initialization

Synchronization can be started without a separate initialization (See the initialization in Chapter 4). 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.

If the sync is done without the initialization, the Alert command(s) (from the client) in Packet #1 is sent within Packet #3, in which the Sync command(s) are also placed. Also, the Alert command(s) (from Server) in Packet #2 is sent within Packet #4, in which the Sync command(s) are also placed.

The sync server MUST be able to handle both the cases; sync with a separate initialization or sync without a separate initialization.

See the example of this in Appendices.

2.11.1 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 may 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 required) to the client. I.e., the client cannot be sure whether it is communicating with the correct server.

2.12 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., Retry Alert), the server MUST assume that the client has stopped the synchronization and start he synchronization from the beginning. The server MUST NOT update its 'Last' sync anchor. The server MUST NOT either update the client Next sync anchor.

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

2.12.1 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 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 '1.1'.
    - The value of the VerProto element MUST be 'SyncML/1.1'.
    - Session ID MUST be included to indicate the ID of a sync session.
    - MsgID MUST be used to unambiguously identify the message belonging 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.

2.12.1.1 Example of Busy Status

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.1</VerDTD>
      <VerProto>SyncML/1.1</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>101</Data> <!--Statuscode for Busy-->
      </Status>
   </SyncBody>
</SyncML>
```

2.12.2 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 '1.1'.
- The value of the VerProto element MUST be 'SyncML/1.1'.
- Session ID MUST be included to indicate the ID of a sync session.
- MsgID MUST be used to unambiguously identify the message belonging 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 is required.
   - 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.

2.12.2.1 Example of Result Alert

```
<SyncML>
  <SyncHdr>
    <VerDTD>1.1</VerDTD>
    <VerProto>SyncML/1.1</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>
```

3 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.

3.1 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 Representation protocol spec). The Chal contains a challenge applicable to the requested resource. The device MAY repeat the request with a suitable Cred element (See Representation protocol). 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 required, a new digest must be 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 required) requirements are:

- The response message indicating the authentication failure on server layer (see chapter 3.3) must contain 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.

3.2 Authorization

The Cred element MUST be included in requests (message or command), which are sent after receiving the 401 or 407 response if the request is repeated. In addition, it can be sent in the first request from a device if the authentication is pre-configured to be required. The content of the Cred

element is specified in [1]. The authentication type is dependent on the challenge (See the previous chapter) or the pre-configuration.

3.3 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.

3.4 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 Representation Protocol.) 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).

3.5 Authentication Examples

3.5.1 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 practise, they would be there.

Pkg #1 from Client

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.1</VerDTD>
      <VerProto>SyncML/1.1</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>1.1</VerDTD>
      <VerProto>SyncML/1.1</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>1.1</VerDTD>
      <VerProto>SyncML/1.1</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>1.1</VerDTD>
      <VerProto>SyncML/1.1</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>
         <Data>212</Data> <!--Authenticated for session-->
      </Status>
...
   </SyncBody>
</SyncML>
```

3.5.2 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 practise, they would be there.

Pkg #1 from Client

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.1</VerDTD>
      <VerProto>SyncML/1.1</VerProto>
      <SessionID>1</SessionID>
      <MsgID>1</MsgID>
      <Target><LocURI>http://www.syncml.org/sync-server</LocURI></Target>
      <Source>
         <LocURI>IMEI:493005100592800</LocURI>
         <LocName>Bruce2</LocName> <!-- userId -->
      </Source>
   </SyncHdr>
   <SyncBody>
...
   </SyncBody>
</SyncML>
```

Pkg #2 from Server

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.1</VerDTD>
      <VerProto>SyncML/1.1</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-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>1.1</VerDTD>
```

```
      <VerProto>SyncML/1.1</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-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>1.1</VerDTD>
      <VerProto>SyncML/1.1</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>
```

4 Sync Initialization

The sync initialization is required that the actual synchronization (See Chapters 5-7), i.e., the sync commands, can also be transmitted and processed. Prior to the sync initialization, the SyncML server may alert the client to trigger synchronization with it (See Chapter 8) 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 are desired to be synchronized and which protocol type is 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 2.7).

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.

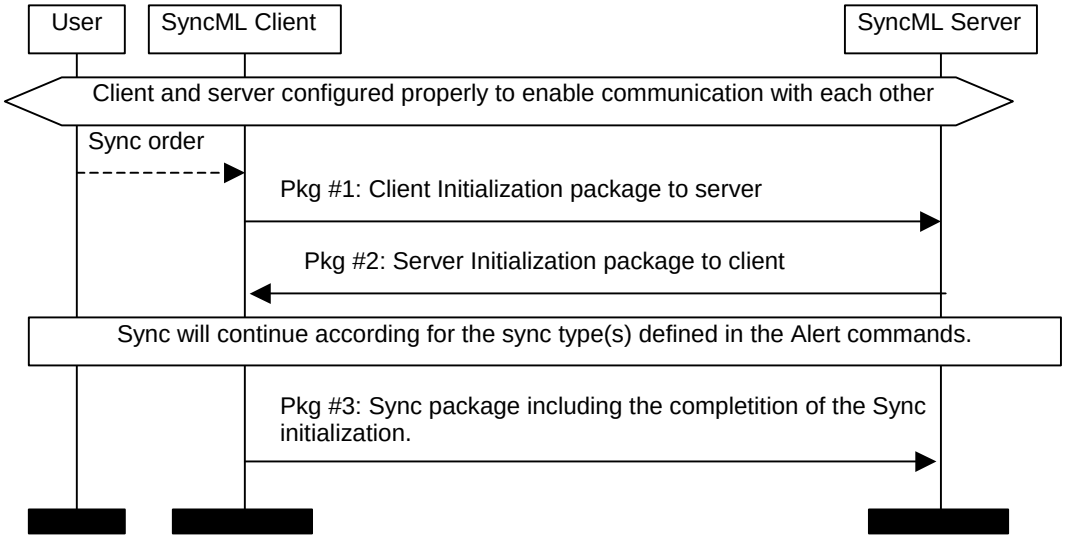

Figure 6 MSC of Synchronization Initialization

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 SyncML session that means that all messages have the same SyncML session ID.

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

4.1 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 6) from the client to the server are:

4. Requirements for the elements within the SyncHdr element.
    - The value of the VerDTD element MUST be '1.1'.
    - The VerProto element MUST be included to specify the sync protocol and the version of the protocol. The value MUST be 'SyncML/1.1' 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 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.
5. The Alert element(s) for each database to be synchronized MUST be included in SyncBody and the following requirements exist.
    - CmdID is required.
    - The response SHOULD be required for 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 2.2.1). The sync anchors are carried inside the Meta element in the Item element.
6. If the service capabilities are sent from the client to the server, the following requirements for the Put command in the SyncBody exist.

- CmdID is required.
- 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 './devinf11'.
- The Data element is used to carry the device and service information data.

7. If the service capabilities are requested from the server, the following requirements for the Get command in the SyncBody exist.
   - CmdID is required.
   - 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 './devinf11'.
8. The Final element MUST be used for the message, which is the last in this package.

4.1.1 Example of Sync Initialization Package from Client

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.1</VerDTD>
      <VerProto>SyncML/1.1</VerProto>
      <SessionID>1</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, which
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>./devinf11</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>
                <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>
                </DataStore>
                <CTCap>
                   <CTType>text/x-vcard</CTType>
                   <PropName>BEGIN</PropName>
                      <ValEnum>VCARD</ValEnum>
                   <PropName>END</PropName>
                      <ValEnum>VCARD</ValEnum>
                   <PropName>VERSION</PropName>
                      <ValEnum>2.1</ValEnum>
                   <PropName>N</PropName>
                   <PropName>TEL</PropName>
                      <ParamName>VOICE</ParamName>
                      <ParamName>CELL</ParamName>
                </CTCap>
                <SyncCap>
                   <SyncType>01</SyncType>
                   <SyncType>02</SyncType>
                </SyncCap>
             </DevInf>
          </Data>
       </Item>
    </Put>
    <Get>
       <CmdID>3</CmdID>
       <Meta><Type xmlns='syncml:metinf'>application/vnd.syncml-devinf+xml</Type></Meta>
       <Item>
          <Target><LocURI>./devinf11</LocURI></Target>
       </Item>
    </Get>
    <Final/>
  </SyncBody>
</SyncML>
```

4.2 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 '1.1'.
- The VerProto element MUST be included to specify the sync protocol and the version of the protocol. The value MUST be 'SyncML/1.1' 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 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 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 2.9).
    - 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 2.2.1).
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 './devinf11'.
    - 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 is required.
    - The response SHOULD be required for 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 2.2.1).
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 is required.
    - 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 './devinf11'.
- 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 is required.
   - 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 './devinf11'.
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.

4.2.1 Example of Sync Initialization Package from Server

```
<SyncML>
  <SyncHdr>
    <VerDTD>1.1</VerDTD>
    <VerProto>SyncML/1.1</VerProto>
    <SessionID>1</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>./devinf11</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>./devinf11</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>
         <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>
         </DataStore>
         <CTCap>
            <CTType>text/x-vcard</CTType>
            <PropName>BEGIN</PropName>
               <ValEnum>VCARD</ValEnum>
            <PropName>END</PropName>
               <ValEnum>VCARD</ValEnum>
            <PropName>VERSION</PropName>
               <ValEnum>2.1</ValEnum>
            <PropName>N</PropName>
            <PropName>TEL</PropName>
               <ParamName>VOICE</ParamName>
               <ParamName>CELL</ParamName>
            <CTType>text/vcard</CTType>
            <PropName>BEGIN</PropName>
               <ValEnum>VCARD</ValEnum>
            <PropName>END</PropName>
               <ValEnum>VCARD</ValEnum>
            <PropName>VERSION</PropName>
               <ValEnum>3.0</ValEnum>
            <PropName>N</PropName>
            <PropName>TEL</PropName>
               <ParamName>VOICE</ParamName>
               <ParamName>FAX</ParamName>
               <ParamName>CELL</ParamName>
            <CTType>text/vcard</CTType>
            <PropName>BEGIN</PropName>
               <ValEnum>VCARD</ValEnum>
            <PropName>END</PropName>
               <ValEnum>VCARD</ValEnum>
            <PropName>VERSION</PropName>
               <ValEnum>3.0</ValEnum>
            <PropName>N</PropName>
            <PropName>TEL</PropName>
               <ParamName>VOICE</ParamName>
               <ParamName>FAX</ParamName>
               <ParamName>CELL</ParamName>
         </CTCap>
         <SyncCap>
```

```
                  <SyncType>01</SyncType>
                  <SyncType>02</SyncType>
                  <SyncType>07</SyncType>
               </SyncCap>
            </DevInf>
         </Data>
      </Item>
   </Results>
   <Alert>
      <CmdID>5</CmdID>
      <Data>201</Data> <!-- 201 = TWO_WAY_ALERT -->
      <Item>
         <Target><LocURI>./dev-contacts</LocURI></Target>
         <Source><LocURI>./contacts/james_bond</LocURI></Source>
         <Meta>
            <Anchor xmlns='syncml:metinf'>
               <Last>200005021T081812Z </Last>
               <Next>200005022T093223Z </Next>
            </Anchor>
         </Meta>
      </Item>
   </Alert>
   <Final/>
  </SyncBody>
</SyncML>
```

4.3 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.

4.3.1 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.

4.3.2 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.

4.3.3 Initialization Failure

If the initialization fails and a defined error code 0 is sent, the devices MUST act according that error type.

5 Two-Way Sync

Two-way sync is a normal synchronization type in which the client and the server are required to 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 7, there is depicted the MSC of the client initiated two-way sync scenario.

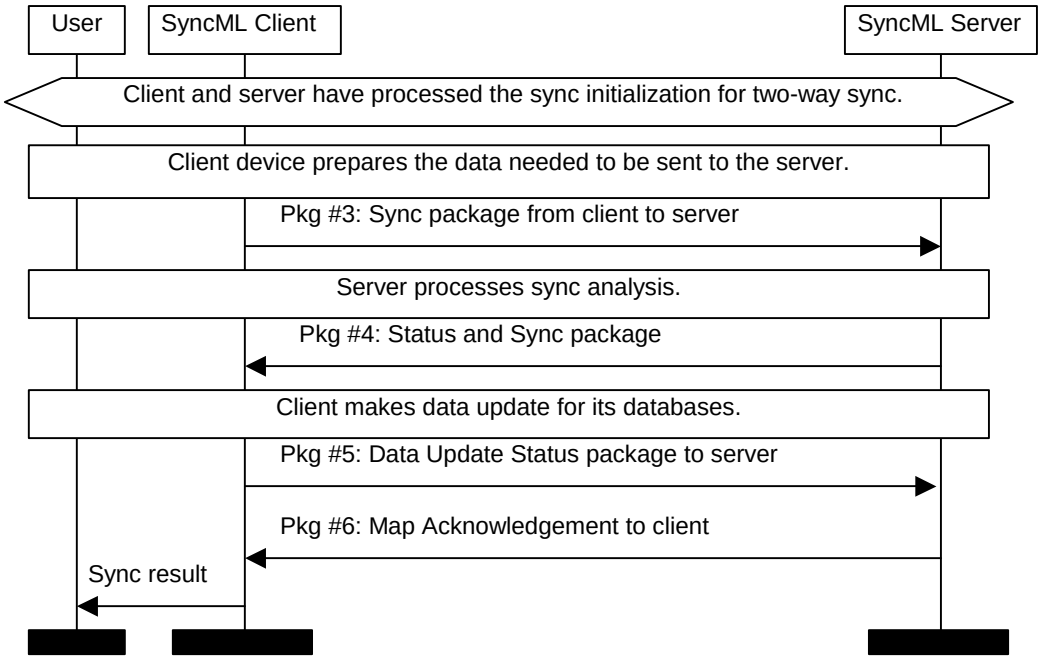

Figure 7 MSC of Two-Way Sync

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 SyncML session that means that all messages have the same SyncML 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 2.10), the number of a package in the figure may not describe the actual atomic number of a package in a synchronization session.

5.1 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 7). 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.

1. Requirements for the elements within the SyncHdr element.
    - The value of the VerDTD element MUST be '1.1'.
    - The VerProto element MUST be included to specify the sync protocol and the version of the protocol. The value MUST be 'SyncML/1.1' 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 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 client if 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 2.2.1).
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 './devinf11'.
    - 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 is required.
    - The response SHOULD be required for the Sync command.
    - Target is used to specify the target database.
    - Source is used to specify 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 2.7). This information can only be sent at the first message belonging this package.

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

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 is required.
   - The response SHOULD be required for 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.

5.1.1 Example of Sending Modifications to Server

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.1</VerDTD>
      <VerProto>SyncML/1.1</VerProto>
      <SessionID>1</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>200005022T093223Z </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>
```

[3] It is not required that the SyncML clients support the Add operation when sending modifications. They may use the Replace operation for additions and then, the receiving device must make addition if the UID of an object does not exist.

```
            <!--Free memory (bytes) in Calendar database on a device -->
            <FreeId>81</FreeId>
            <!--Number of free records in Calendar database-->
          </Mem>
        </Meta>
        <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>
```

5.2 Server Modifications to Client

The sync package (Refer Pkg #4 in Figure 7) 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 '1.1'.
   - The value of the VerProto element MUST be 'SyncML/1.1'.
   - Session ID MUST be included to indicate the ID of a sync session.
   - MsgID MUST be used to unambiguously identify the message belonging 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 is required.
   - The response can be required for the Sync command. (See the Caching of Map Item, Chapter 2.3.1)

- Target is used to specify the target database.
- Source is used to specify the source database.

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 is required.
   - The response can 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.

5.2.1 Example of Sending Modifications to Client

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.1</VerDTD>
      <VerProto>SyncML/1.1</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>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>
```

[4] It is not required that the devices support the Add operation. They may use the Replace operation for additions and then, the receiving device must make addition if the UID of an object does not exist.

```
        <CmdID>4</CmdID>
        <Target><LocURI>./dev-contacts</LocURI></Target>
        <Source><LocURI>./contacts/james_bond</LocURI></Source>
        <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 place here.--></Data>
           </Item>
        </Add>
     </Sync>
     <Final/>
  </SyncBody>
</SyncML>
```

5.3 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 2.3.1). 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 '1.1'.
    - The value of the VerProto element MUST be 'SyncML/1.1'.
    - Session ID MUST be included to indicate the ID of a sync session.
    - MsgID MUST be used to unambiguously identify the message belonging 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 is required.
    - The Source and Target elements are required in the Map element.
    - The response is required to the Map operation.
    - The client has to 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.

5.3.1 Example of Data Update Status to Server

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.1</VerDTD>
      <VerProto>SyncML/1.1</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>
      <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>
```

5.4 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 '1.1'.
    - The value of the VerProto element MUST be 'SyncML/1.1'.
    - Session ID MUST be included to indicate the ID of a sync session.
    - MsgID MUST be used to unambiguously identify the message belonging 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.
    - The response MUST NOT be required for this message.
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.

5.4.1 Example of Map Acknowledge

```
<SyncML>
   <SyncHdr>
      <VerDTD>1.1</VerDTD>
      <VerProto>SyncML/1.1</VerProto>
      <SessionID>1</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>
```

5.5 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 practise, 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 2.10, 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.

5.6 Error Case Behaviors

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

5.6.1 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.

5.6.2 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.

5.6.3 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.

5.6.4 Errors with Defined Error Codes

If the device receives a defined error code 0, it MUST act according that error type.

6 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 8, there is depicted the MSC for this scenario.

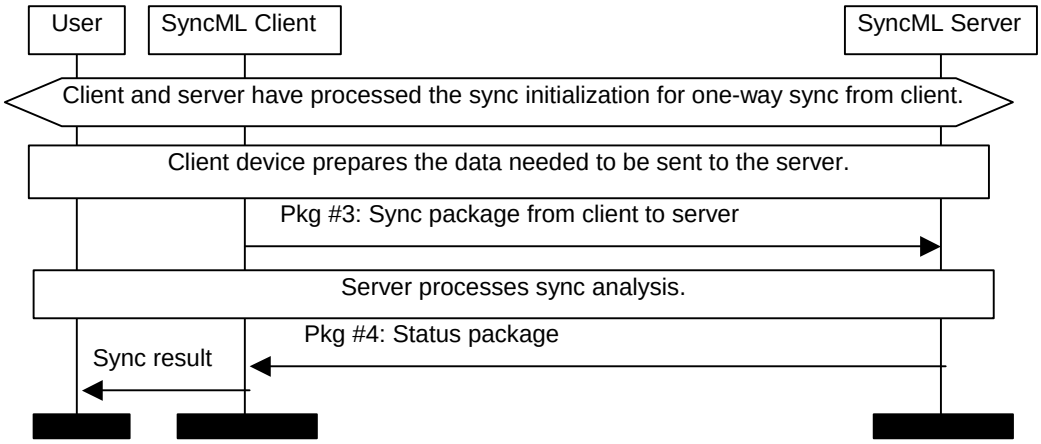

Figure 8 MSC of One-Way Sync from Client only

The package flow presented above is one SyncML session that means that all messages have the same SyncML 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 2.10), the number of a package in the figure may not describe the actual atomic number of a package in a synchronization session.

6.1 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 8). 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 5.1.

6.2 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.

[5] 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 '1.1'.
    - The value of the VerProto element MUST be 'SyncML/1.1'.
    - Session ID MUST be included to indicate the ID of a sync session.
    - MsgID MUST be used to unambiguously identify the message belonging 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 #1 is completely received.

6.3 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 required. The Alert code is 203.
2. In Package #3, the Sync element (Pkg #3) from the client to the server is required to include all data from the source database (client database).

6.4 Error Cases Behavior

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

6.4.1 No Packages from Server after Initialization

See Chapter 5.6.1.

6.4.2 Errors with Defined Error Codes

See Chapter 5.6.4.

7 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 9, there is depicted the MSC for this scenario.

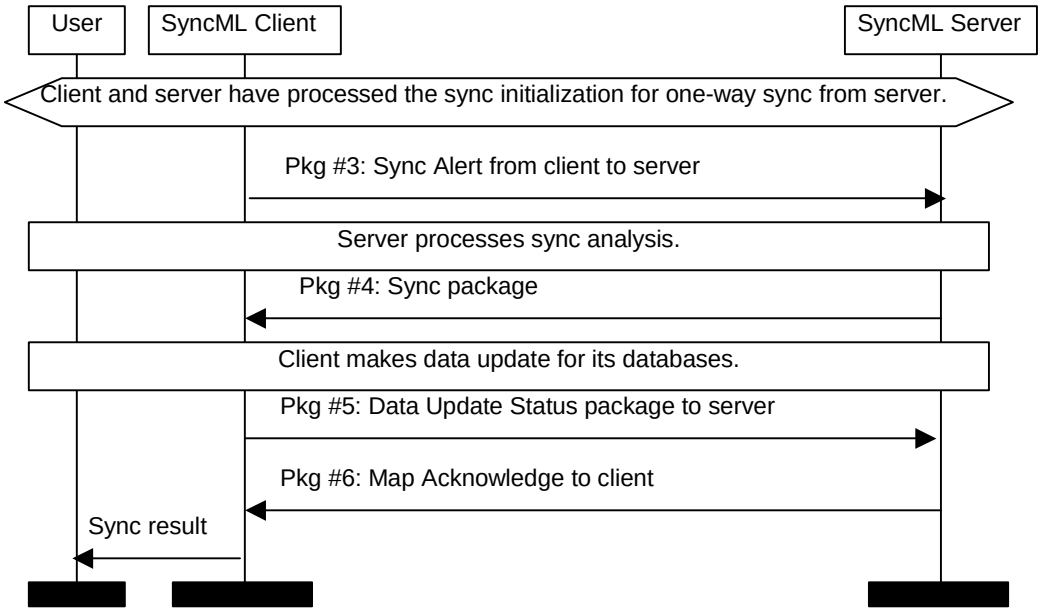

Figure 9 MSC of Sync from Server Only

The package flow presented above is one SyncML session that means that all messages have the same SyncML 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 2.10), the number of a package in the figure may not describe the actual atomic number of a package in a synchronization session.

7.1 Sync Alert to Server

The sync package (Pkg #3 in Figure 9) 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 5.1 is:

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

7.2 Server Modifications to Client

See Chapter 5.2.

7.3 Data Update Status from Client

See Chapter 5.3.

7.4 Map Acknowledge from Server

See Chapter 5.4.

7.5 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. I.e., 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 7.1), the value for the Alert code is 205.
2. In the Server Modifications package to the client (See Chapter 7.2), the Sync element is required to 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.

7.6 Error Cases

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

7.6.1 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 SyncML session and try to get the modifications later by starting the sync from the beginning.

7.6.2 No Data Update Status from Client

See Chapter 5.6.2.

7.6.3 No Map Ack from Server

See Chapter 5.6.3.

7.6.4 Errors with Defined Error Codes

See Chapter 5.6.4.

8 Server Alerted Sync

This sync case is intended to provide a possibility for the server to alert the client to perform sync. That is, the server informs the client to starts sync with the server. When the server alerts the client, it also tells it which type of sync is initiated. Figure 10 shows the MSC, how sync is alerted by the server.

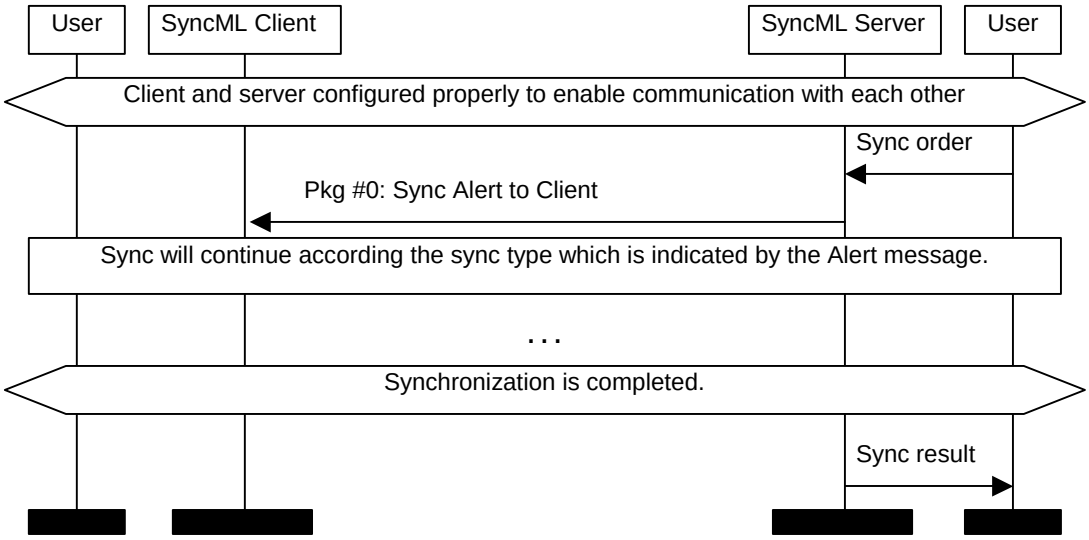

Figure 10 MSC of Server Alerted Sync

In the server alerted sync, the sent packages are the same as in any sync types except the alert message, which is sent from the server to client.

The package flow presented above is one SyncML session that means that all messages have the same SyncML session ID. The same Session ID used here MUST also be used at the synchronization initialization.

8.1 Sync Alert

The sync alert is sent from the server when the server wants the client to start synchronization. This message MUST indicate which type of sync the server wants. The requirements for the elements within this sync alert package are:

1. Requirements for the elements within the SyncHdr element.
    - The value of the VerDTD element MUST be '1.1'.
    - The value of the VerProto element MUST be 'SyncML/1.1'.
    - Session ID MUST be included to indicate the ID of a sync session.
    - MsgID MUST be used to unambiguously identify the message belonging a sync session and traveling from the server to the client.

- The Target element MUST be used to identify the target device. The character '/' MUST be used when there is no sync relationship between the server and the client.
- 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 MUST be included in SyncBody.
   - CmdID is required.
   - The response SHOULD be required for the Alert command.
   - The Item element SHOULD include the target database. Note, if this is the alert for the first time sync, the target database may not be included but the sync client determines it according the meta information sent within the alert.
   - The Item element MUST BE used to specify the source database.
   - 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 (e.g., MIME type) to be synchronized.
   - The Data element is used to include the Alert code. The alert code is one of the alert codes used by the server (Values 206-210.). See the alert codes in Alert Codes.
3. Final MUST be used for the message, which is the last in this package.

When the client receives this message, it continues according to the sync type indicated by the Alert element. The status element is also included in the first package from the client to the server. If the error occurs, the error status is returned and the defined error codes are used.

8.2 Error Cases Behavior

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

8.2.1 No Packages from Client

If the server has sent an alert to the client and it does not get any complete response to, the server SHOULD try to alert the client later.

8.2.2 Errors with Defined Error Codes

See Chapter 5.6.4.

9 Terminology

9.1 Definitions

Client Modification – A modification of an item, which occurs in a client database before the modification is synchronized to the server database.

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 (also see LUID).

LUID (Locally Unique Identifier) – A number assigned to an object in a database. LUID values are only unique locally, i.e., to a particular SyncML client database, but MAY be present on other SyncML client databases. In this protocol, the SyncML client device assigns to each object a locally unique, non-reusable identifier, or LUID. They are unique per device and per application.

Request – A message or a command sent from a device to another.

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 Engine – The portion of a SyncML server that can analyze a data set and modifications to that data set made by both SyncML server and SyncML client. The synchronization engine will implement policies to enable the detection and resolution of conflicting changes.

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.

9.2 Abbreviations

DTD	Document Type Definition
GUID	Global Unique IDentifier
HTTP	HyperText Transfer Protocol
IMEI	International Mobile Equipment Identifier
LUID	Local Unique Identifier
MSC	Message Sequence Chart
MSG	Message
OBEX	OBject Exchange protocol

WSP	Wireless Session Protocol
XML	Extensible Markup Language

10 References

[1] SyncML Representation Protocol, Data Synchronization Usage v1.1, SyncML.

[2] SyncML Reference Toolkit Manual, SyncML.

[3] SyncML Meta Information Specification and DTD v1.1, SyncML.

[4] SyncML Device Information Specification and DTD v1.1, SyncML.

[5] The Internet Mail Consortium, vCalendar - The Electronic Calendaring and Scheduling Exchange Format, Version 1.0, September 1996, IETF.

11 Appendices

11.1 Protocol Values

Here are listed all protocol values (string values), which can be used in the VerProto element. The protocol version 1.1 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.

11.2 Alert Codes

Here are listed all Alert codes and values, which are used in the Type element of Meta 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.
224-250	-	Reserved for future SyncML usage.

11.3 Conformance Requirements

This section defines static conformance requirements for SyncML servers and client conforming to this specification. Also, the requirements for supporting SyncML commands are defined.

11.3.1 Conformance Requirements for SyncML Server

Table 2 Sync type conformance requirements for devices acting as SyncML server

Sync Type	**Reference**	**Status**
Support of 'two-way sync'	Chapter 4	MUST
Support of 'slow two-way sync'	Chapter 5.5	MUST
Support of 'one-way sync from client only'	Chapter 6	MAY
Support of 'refresh sync from client only'	Chapter 6.3	MAY
Support of 'one-way sync from server only'	Chapter 7	MAY
Support of 'refresh sync from server only'	Chapter 7.5	MAY
Support of 'sync alert'	Chapter 8	MAY
Support of ‘busy signalling’	Chapter 2.12	SHOULD

11.3.2 Conformance Requirements for SyncML Client

Table 3 Sync type conformance requirements for devices acting as SyncML client

Sync Type	**Reference**	**Status**
Support of 'two-way sync'	Chapter 4	MUST
Support of 'slow two-way sync'	Chapter 5.5	MUST
Support of 'one-way sync from client only'	Chapter 6	MAY
Support of 'refresh sync from client only'	Chapter 6.3	MAY

Support of 'one-way sync from server only'	Chapter 7	MAY
Support of 'refresh sync from server only'	Chapter 7.5	MAY
Support of 'sync alert'	Chapter 8	MAY

11.4 Examples

11.4.1 WBXML Example

Here is an example of Package #3 (as defined in) 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 1D ”-” ”/” ”/” ”S” ”Y” ”N” ”C” ”M” ”L” ”/” ”/” ”D” ”T” ”D” ” ” ”S” ”y” ”n” ”c”
”M” ”L” ” “ ”1” ”.” ”1” ”/” ”/” ”E” ”N” 6D 6C 71 C3 03 "1" "." "1" 01 72 C3 0A "S" "y" "n"
"c" "M" "L" "/" "1" "." "1" 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 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
1D	String table length
”-” ”/” ”/” ”S” ”Y” ”N” ”C” ”M” ”L” ”/” ”/” ”D” ”T” ”D” ” ” ”S” ”y” ”n” ”c” ”M” ”L” ” “ ”1” ”.” ”1” ”/” ”/” ”E” ”N”	-//SYNCML//DTD SyncML 1.1//EN
6D	<SyncML>
6C	<SyncHdr>
71	<VerDTD>
C3	Opaque data follows
03	Length of opaque data
"1" "." "1"	String '1.1'
01	</VerDTD>
72	<VerProto>
C3	Opaque data follows
0A	Length of opaque data
"S" "y" "n" "c" "M" "L" "/" "1" "." "1"	String 'SyncML/1.1'
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>
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>

11.4.2 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>1.1</VerDTD>
      <VerProto>SyncML/1.1</VerProto>
```

```
      <SessionID>1</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>
         <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>1.1</VerDTD>
      <VerProto>SyncML/1.1</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>
      <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>200005021T081812Z </Last>
            <Next>200005022T093223Z </Next>
          </Anchor>
        </Meta>
      </Item>
    </Alert>
    <Sync>
      <CmdID>6</CmdID>
      <Target><LocURI>./dev-contacts</LocURI></Target>
      <Source><LocURI>./contacts/james_bond</LocURI></Source>
      <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>
```