Change Document for SyncML Synchronization Protocol

Specification version: 1.0.1

Specification date: 2000-06-15

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

Copyright Notice

1 Formatting 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 interoperated as described in [RFC 2119].

1.1 Errata Type Classifications

The errata types are classified according to the following scheme:

CLARIFICATION: Textual enhancement that provides a clearer explanation of a specification item without changing any behavior.

CORRECTION: A modification that obsoletes some items in the current published specification.

PROBLEM: A known problem for which an erratum has yet to be proposed.

2 Errata

2.1 Alert 222

2.1.1 Problem

It is unclear when a client should send the Alert 222 (Next Message).

2.1.2 Solution

in section 2.9, Multiple Messages in Package, change: "can be omitted" to: "MAY be omitted"

2.1.3 Other specifications/erratas affected

None.

2.2 Discrepancies in the Sync Anchor Example

2.2.1 Problem

Section 2.2.1.1 has discrepancies between dates in text and diagram.

2.2.2 Solution

Reword 2nd and 3rd Paragraph of section 2.2.1.1 to:

"The sync session #1 is started at 10:10:10 AM on the 10 th of October 2001. The previous synchronization (before the sync session #1) was started at 09:09:09 AM on the 9 th 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 9 th of September, 2001).

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

2.2.3 Other specifications/erratas affected

None.

2.3 Status in Authentication Failure

2.3.1 Problem

The documentation is does not clearly state that when returning authentication failure status on the server layer that status must be returned for each command in the request.

2.3.2 Solution

Add the following sentence to the first bullet at the end of section 3.1. “A status command MUST be provided for every command received in the request.”

2.3.3 Other specifications/erratas affected

None.

2.4 Sync Anchors Example

2.4.1 Problem

The concurrent example does not follow the specified definition of “Last”

Chapter 2.2.1.1 Example of Database Sync Anchor Usage, Figure 1 Example of Sync Anchor Usage:

Pkg #1: Last ('Empty'), Next(20011111T111111Z)

The DTD definition says that the value MUST specify either an UTC based ISO 8601 [1] date/time stamp or a monotonically increasing numeric integer string. If a date/time stamp, then the text MUST be in the complete representation, basic format defined by [ISO8601].

2.4.2 Solution

Change “Last” value to time 0 (zero). Change the current example to:

Pkg #1: Last (00000000T000000Z), Next(20011111T111111Z)

2.4.3 Other specifications/erratas affected

None.

2.5 Busy Signalling

2.5.1 Problem

The current definition on how an overloaded server will send busy signal to the client so that the client should wait and continue the synchronization later, is, in some cases not a solution in the server, if the bottleneck in the server is the connection to the database.

If the server is busy, then the server must parse the SyncML document and store it in the database.

2.5.2 Solution

Chapter 2.11 Busy Signaling:

Change the requirement on the server from MUST to SHOULD.

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 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.

2.5.3 Other specifications/erratas affected

None.

2.6 Busy Signalling Conformance Requirements

2.6.1 Problem

The current specification defines the requirements on Server Busy in the text but it is not a "Conformance Requirement" table in the end of the document.

2.6.2 Solution

Add a new Conformance Requirement table in chapter 11.3:

Support of 'busy signalling'	Chapter 2.11	SHOULD

2.6.3 Other specifications/erratas affected

None.

2.7 Server Alerted Sync

2.7.1 Problem

Chapter 8 of the SyncML Sync Protocol Document 1.0.1 contains some ambiguities and possible errors.

Bullet point 5 of section 1 (SyncHdr) of 8.1 Sync Alert states that "The Target element MUST be used to identify the target device.". The server may not know this if there is no sync relationship between server and client.

Section 8.2.1 states that if a server "does not get any complete response to" an alert message, "The server MUST try to alert the client later.". This clearly should not happen in certain cases.

2.7.2 Solution

Bullet number 5:

Change text to "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."

Section 8.2.1:

Change the text of to: "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."

2.7.3 Other specifications/erratas affected

None.

3 Enhancements

3.1 Large Object Delivery

3.1.1 Abstract

A single SyncML message is limited in size by the underlying PDU of the transport it uses. SyncML provides no mechanism to split data payload across multiple messages. When working over a wireless link using a transport such as WSP this imposes a limitation that is likely to have real world impact on the size of objects that can be synchronised. This document proposes a solution that enables objects to be segmented and transmitted across multiple SyncML messages.

3.1.2 Added wording to Synchronization Protocol

Add a new paragraph after 2.9 to specify the large object delivery:

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>
...
```

Add also a new entry to the table in section 11.2 to specify the Alert code ‘223’.

223	END OF DATA FOR CHUNKED OBJECT NOT RECEIVED	Specifies a request for the recipient to inform the originator that a chunked data object was not completely received.

4 References

[RFC 2119] Key words for use in RFCs to Indicate Requirement Levels, IETF.