SyncML Device Management Protocol

Version 1.1

Abstract

This document specifies the SyncML Device Management Protocol. This protocol is used to transfer management actions between the client and the management server.

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**
V0.8	2001-11-21	Alpha release to supporters. Alpha release contains protocol and representation specifications in one document.
V1.1.0	2002-02-15	Release for version 1.1

Copyright Notice

Table of Contents

1 Introduction

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

2 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 interpreted as described in "Key words for use in RFCs to Indicate Requirement Levels" [RFC 2119].

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

3 Terminology

See SyncML Representation Protocol [REPPRO] and SyncML Synchronization Protocol [SYNCPRO] for definitions of SyncML terms used within this specification.

Full device URI

Full path to a device resource specified in the device's context. It is always URI relative to devices root management object. Full device URI must always be used in the present specification.

Interior management object

Management object having only children management object and no value associated with it.

Leaf management object

End of the management object tree; management object having value and no children management objects.

Message

Atomic unit in SyncML DM Protocol protocol, one packet that the bearer network is able to accept. One SyncML DM Protocol package could be divided into many messages.

Package

Package is a conceptual set of commands that could be spread over multiple messages.

Resource

A network data object or service that can be identified by a URI, as defined in Section 3.2 of Hypertext Transfer Protocol [RFC 2616] Resources may be available in multiple representations (e.g. multiple languages, data formats, size, and resolutions) or vary in other ways.

4 SyncML Device Management Protocol Overview

SyncML Device Management Protocol allows management commands to be executed on management objects and it uses a package format similar SyncML Synchronization Protocol [SYNCPRO] and SyncML Representation Protocol [REPPRO]. A management object might reflect a set of configuration parameters for a device. Actions that can be taken against this object might include reading and setting parameter keys and values. Another management object might be the run-time environment for software applications on a device. Actions that can be taken against this type of object might include installing, upgrading, or uninstalling software elements.

Actions are represented by SyncML Device Management Protocol Commands, which are described in SyncML Representation Protocol, Device Management Usage [DMREPU]. The commands and message structure used correspond identically to that of the [SYNCPRO]. Thus, the DTD for the Management Protocol is the DTD from [SYNCPRO].

5 Management object addressing

Each management object MUST be addressed by a unique full device URI. URI MUST follow requirements specified in Uniform Resource Identifiers (URI) [RFC 2396]. All URI's are case sensitive in SyncML DM Protocol. Management Object addressing is defined in SyncML Device Management Tree and Descriptions [DMTND].

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

In SyncML DM Protocol commands target and source of command is refered with `Target` and `Source` elements correspondingly. `Target` element refers to managed client and `Source` refers to device management server. Exceptions to this approach are mentioned in management command requiring exception.

6 Multible messages in package

SyncML DM Protocol utilizes multible message in package functionality specified in [SYNCPRO] Section 2.9. `Alert` code 1222 MUST be used to ask for more messages belonging to package.

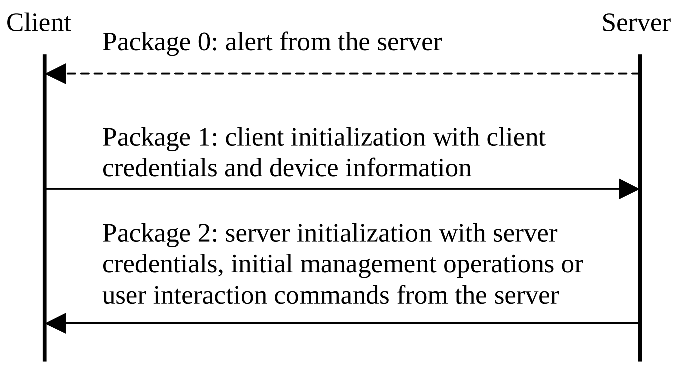
7 Large Object Handling

With SyncML DM Protocol large objects which do not fit into one underlying bearer package can be transferred with multible SyncML DM Protocol packages. For this SyncML DM Protocol utilizes Large Object Handling functionality specified in [SYNCPRO] Section 2.10.

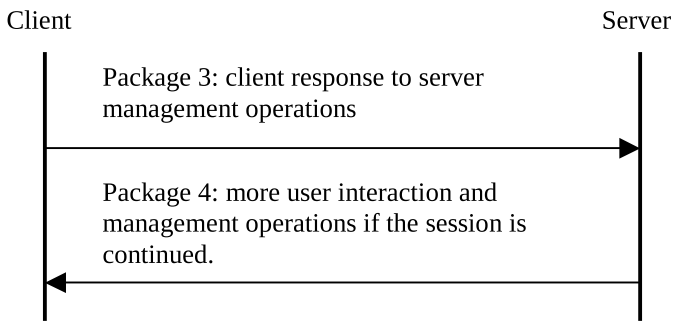
8 SyncML DM Protocol packages

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

The following chart depicts the two phases.


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

Setup phase


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

Management phase

Management Phase consist of a number of protocol iterations[1]. Content of package sent from server to client determines whether the session must be continued or not. If the server sends management operations in package that need responses (`Status` or `Results`) from the client, the management phase of the protocol continues with new package from

[1] Protocol iteration means a package from client to server and a package from server to client.

client to server containing client responses to management operations. Response package from client starts new protocol iteration. The server can send new management operation package and therefore initiate new protocol iteration as many times as it wishes.

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

Handling of packages can contain operations consuming unpredictable amount of time. Therefore SyncML DM Protocol do not specify any timeouts between packages.

If not otherwise stated by management commands, the client and server may freely choose the execution order of the management commands sent in the package. However, when execution order is required by the parent management command, commands MUST be executed in the order they were sent.

Client and server may decide to abort the session at any time. Reasons for session abort may be server shutdown, client power-down, user interaction on the client, etc. In this case the aborting party SHOULD send a SESSION ABORT `Alert`. The message MUST also include `Status` messages of all the management commands that the aborting party executed before the abort operation. The response of the receiving party is ignored (although the aborting party MAY want to fully receive it if the bearer makes it necessary to wait for the transaction result) and the connection is closed.

Some cases of session aborts are not controllable, for example if the client goes out of coverage or its battery runs down. Servers and clients must be prepared for non-signalled session aborts as well.

Client MUST NOT send any management actions other than `Replace` command containing DevInfo to the server.

8.1 Package 0: Management Initiation Alert from server to client

Many devices cannot continuously listen for connections from a management server. Other devices simply do not wish to “open a port” (i.e. accept connections) for security reasons. However, most devices can receive unsolicited messages, sometimes called “notifications”.

A management server can use this notification capability to cause the client to initiate a connection back to the management server. SyncML DM Protocol 1.1 specifies several Management Initiation notification bearers. Definition of bearers and notification content can be found from [DMNOTI] specification.

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

8.2 Package 1: Initialization from client to server

The setup phase is virtually identical to that described in the [SYNCPRO], Section 4. The purpose of the initialization package sent by the client is to:

- To send the device information (like manufacturer, model etc) to a Device Management Server as specified [DMSTDOBJ]. Client MUST send device information in the first message of management session.
- Identify the client to the server according to the rules specified in Section 9.
- To inform the server whether the management session was initiated by the server (by sending a trigger in Package 0) or by the client (like end user selecting a menu item).

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

1. The requirements for the elements within the `SyncHdr` element.
    - The value of the `VerDTD` element MUST be '1.1'.
    - The value of the `VerProto` element MUST be 'DM/1.1'.
    - `SessionID` MUST be included to indicate the ID of the management session. If the client is responding to notification, with alert code SERVER-INITIATED MGMT (1200), then `SessionID` MUST be same as in notification. Otherwise client generates `SessionID` which SHOULD be unique `SessionID` in client. Same `SessionID` MUST be used throughout the whole session.
    - `MsgID` MUST be used to unambiguously identify the message belonging to the management session from server to client.
    - The `Target` element MUST be used to identify the target device.
    - The `Source` element MUST be used to identify the source device.
    - The `Cred` element MAY be included in the authentication message from the Device Management client to Device Management server as specified in Section 9.
2. `Alert` MUST be sent whether the client or the server initiated the management session in the `SyncBody`. The requirement for the `Alert` command follows:
    - `CmdID` is REQUIRED
    - The `Data` element is used to carry the management session type which can be either SERVER-INITIATED MGMT (1200) or CLIENT-INITIATED MGMT (1201)

3. The device information MUST be sent using the `Replace` command in the `SyncBody`. The requirement for the `Replace` command follows:

   - `CmdID` is REQUIRED.
   - An Item element per object found from device information tree. Possible objects in device information tree are specified in [DMSTDOBJ].
   - The `Source` element in the `Item` element MUST have a value indicating URI of object.
   - The `Data` element is used to carry the device information data.

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

8.3 Package 2: Initialization from server to client

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

- Identify the server to the client according to the rules specified in Section 9.
- Optionally server can send user interaction commands.
- Optionally to send management data and commands.

Package 2 MAY close the management session.

The detailed requirements for package 2 are:

1. The 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 ‘DM/1.1’ when complying with this specification.
   - `SessionID` MUST be included to indicate the ID of the management session.
   - `MsgID` MUST be used to unambiguously identify the message belonging to the management session from server to client.
   - The `Target` element MUST be used to identify the target device.
   - The `Source` element MUST be used to identify the source device.

- `Cred` element MAY be included in the authentication message according to the rules described in Section 9. Server is always authenticated to the device but this authentication MAY be accomplished at the transport level.

2. The `Status` MUST be returned in the `SyncBody` for the `SyncHdr` and `Alert` command sent by the client.

3. Any management operation including user interaction in the SyncML document (e.g. `Alert`, `Sequence`, `Replace`) placed into the `SyncBody`.

   - `CmdID` is REQUIRED.
   - `Source` MUST be used if URI is needed to further address the source dataset.
   - `Target` MUST be used if URI is needed to further address the target dataset.
   - The `Data` element inside Item is used to include the data itself if the operation is not a deletion.
   - The `Meta` element inside an operation or inside an `Item` MUST be used. 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).

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

8.4 Package 3: Client response sent to server

The content of package 3 is:

- Results of management actions sent from server to client
- Results of user interaction commands

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

The detailed requirements for package 3 are:

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

   - The value of the `VerDTD` element MUST be '1.1'.
   - The value of the `VerProto` element MUST be ‘DM/1.1’.
   - `SessionID` MUST be included to indicate the ID of the management session.
   - `MsgID` MUST be used to unambiguously identify the message belonging to the management session from server to client.
   - The `Target` element MUST be used to identify the target device.

- The `Source` element MUST be used to identify the source device.
- A `Status` MUST be returned to indicate whether the authentication of device management server was successful or not. The `CmdRef` used in this `Status` element MUST be `SyncHdr`.

2. `Status` MUST be returned for the `SyncHrd` and `Alert` command sent by the device management server in the `SyncBody`.
3. `Status` MUST be returned in the `SyncBody` for management operations sent by the server in Package 2.
4. `Results` MUST be returned in the `SyncBody` for successfull `Get` operations sent by the server in Package 2. `Results` MUST contain `Meta` element with `Type` and `Format` elements describing content of `Data` element.
5. The `Final` element MUST be used in the `SyncBody` for the message, which is the last in this package.

8.5 Package 4: Further server management operations

Package 4 is used to close the management session. If the server sends any operation in Package 4 that needs response from the client, the protocol restarts from Package 3 with a new protocol iteration.

The detailed requirements for package 4 are:

1. The requirements for the elements within the `SyncHdr` element.
    - The value of the `VerDTD` element MUST be '1.1'.
    - The value of the `VerProto` element MUST be ‘DM/1.1’.
    - `SessionID` MUST be included to indicate the ID of the management session.
    - `MsgID` MUST be used to unambiguously identify the message belonging to the management session from server to client.
    - The `Target` element MUST be used to identify the target device.
    - The `Source` element MUST be used to identify the source device.
2. `Status` MUST be returned for the `SyncHrd` sent by the device management server in the `SyncBody`.
3. Any management operation including user interaction in the SyncML document (e.g. `Alert`, `Sequence`, `Replace`) placed into the `SyncBody`.

- `CmdID` is REQUIRED.
- `Source` MUST be used if URI is needed to further address the source dataset.
- `Target` MUST be used if URI is needed to further address the target dataset.
- The `Data` element inside `Item` is used to include the data itself if the operation is not a deletion.
- The `Meta` element inside an operation or inside an `Item` MUST be used. 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).

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

9 Authentication

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

Both SyncML DM Protocol client and server MUST be authenticated to each other. Authentication can be performed at different levels, however. If the transport level has built-in authentication mechanism then SyncML DM Protocol-level authentication MAY NOT be used. If the transport level does not have sufficiently strong authentication feature, SyncML DM Protocol-level authentication MUST be used. Server and client can both challenge each other if no credentials were given in the original request or the credentials were considered too weak.

It is assumed that SyncML DM Protocol will often be used on top of a transport protocol that offers session-level authentication so that authentication credentials are exchanged only at the beginning of the session (like in TLS or WTLS). If the transport level is not able to provide session authentication, however, each request MUST be authenticated.

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

10 User interaction commands

10.1 Introduction

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

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

10.2 User interaction alert types

These `Alert`'s can be sent only from the server to the terminal. If sent by the terminal, they are ignored by the server. Multiple user interaction `Alert`'s can be present in Package 2, in this case the terminal executes them by arbitrary order (unless `Sequence` is used) and sends back the results in multiple `Status` packages in Package 3. If the protocol continues after Package 4, Package 4 can also contain user interaction `Alert`'s.

When an user interaction is executed, server is notified about the outcome of the interaction in a `Status` message. The user interaction-specific `Status` responses are the following:

- User response to the user interaction.
- `(214) Operation Cancelled` status code if the user cancelled the user interaction.
- `(408) Request timeout` status code if the user didn't respond within the timeout period.
- `(406) Optional feature not supported` status code if the optional user interaction is not implemented.
- `(416) Requested range not satisfiable` status code if the client is not able to display the user interaction because of a device limitation (like too long choice).

Beside the status codes mentioned above, user interaction commands MAY return other status codes described in [REPPRO].

In any case, the server decides to continue or abort the management protocol after having analyzed the returned status code.

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

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

10.2.1 Display

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

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

Example:

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

10.2.2 Confirmation

Confirmation is a binary decision: the user either approves or rejects the option. A new Alert type is introduced for this purpose, the CONFIRM_OR_REJECT. When the terminal receives this Alert, it displays the Alert text then enables the user to select "Yes" or "No". If the answer is "Yes", the processing of the package continues without change in processing.

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

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

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

The `Alert` contains two `Item`'s.

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

Example:

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

Result if user responds "No":

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

10.2.3 User input

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

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

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

Example:

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

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

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

10.2.4 User choice

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

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

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

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

Example for a single-choice `Alert`:

```
<Alert>
  <CmdID>2</CmdID>
  <Data>1103</Data>
  <Item><Data>MDT=10</Data></Item>
  <Item>
    <Data>Select service to configure</Data>
  </Item>
  <Item>
    <Data>CNN</Data>
  </Item>
  <Item>
```

```
      <Data>Mobilbank</Data>
   </Item>
   <Item>
      <Data>Game Channel</Data>
   </Item>
</Alert>
```

Response to this `Alert` returns the selected item.

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

Example to multiple-choice `Alert`

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

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

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

10.2.5 Progress notification (object download)

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

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

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

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

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

10.3 User interaction options

`Alert`'s MAY have optional User interaction parameters in the first `Item`. Optional parameters are represented as one text string inside the `Data` element. If the User interaction `Alert` does not have optional parameters, the first `Item` is empty. The optional parameter string conforms to the URL encoding format specified in [RFC 2396].

The following example uses two optional parameters:

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

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

For the moment following optional parameters are defined.

10.3.1 MINDT (Minimum Display Time)

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

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

Example:

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

10.3.2 MAXDT (Maximum Display Time)

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

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

Example:

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

10.3.3 DR (Default Response)

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

- If the user interaction is Notification, this optional parameter is ignored.
- If the user interaction is a confirmation, 0 means that the reject user interface element is highlighted by default, 1 means that the accept user interface element is highlighted by

default. Highlighted user interface element means that the "default" user interaction (like pressing Enter button) will select the highlighted user interface element. If the client user interface has no notion of highlighted user interface element, this parameter MAY be ignored.

- If the user interaction is user input, DR value specifies the original text in the text input user interface element. This text MUST conform to the optional parameter syntax rules.
- If the user interaction is single-choice, the DR value is the originally highlighted choice item. item e.g. value between 1 and the number of items in the selection list.
- If the user interaction is a multi-choice, the DR value is a minus sign-separated list of originally highlighted values (for example: 2-3).

Examples:

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

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

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

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

10.3.4 MAXLEN (Maximum length of user input)

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

Example:

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

10.3.5 IT (Input Type)

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

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

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

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

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

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

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

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

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

Example:

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

`Status` message delivered to server as response

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

10.3.6 ET (Echo Type)

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

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

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

Example:

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

11 Static conformance requirements

Conformance requirements for SyncML Device Management Protocol are defined in SyncML Device Management Conformance Requirements [DMCONF].

12 References

[RFC 2616]	Hypertext Transfer Protocol -- HTTP/1.1, IETF.
[RFC 2119]	Key words for use in RFCs to Indicate Requirement Levels, IETF.
[RFC 2396]	Uniform Resource Identifiers (URI): Generic Syntax, IETF.
[REPPRO]	SyncML Representation Protocol, version 1.1, SyncML.
[SYNCPRO]	SyncML Synchronization Protocol, version 1.1, SyncML.
[METINF]	SyncML Meta-Information DTD, version 1.1, SyncML.
[DMREPU]	SyncML Representation Protocol, Device Management Usage, version 1.1, SyncML.
[DMNOTI]	SyncML Notification Initiated Session, version 1.1, SyncML.
[DMTND]	SyncML Device Management Tree and Descriptions, version 1.1, SyncML.
[DMSTDOBJ]	SyncML Device Management Standardised Objects, version 1.1, SyncML.
[DMCONF]	SyncML Device Management Conformance Requirements, version 1.1, SyncML.
[DMSEC]	SyncML Device Management Security, version 1.1, SyncML.

13 Appendices

13.1 Protocol Values

VerProto Codes	**Description**
DM/1.1	Indicates that this SyncML message uses the SyncML Device Management Protocol defined by the SyncML Initiative.

14 Protocol examples

In this section several protocol scenarios will be demonstrated.

14.1 One-step protocol initiated by the server

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

14.1.1 Package 1: Initialization from client to server

```
<SyncML xmlns='SYNCML:SYNCML1.1'>
  <SyncHdr>
    <VerDTD>1.1</VerDTD>
    <VerProto>DM/1.1</VerProto>
    <SessionID>1</SessionID>
    <MsgID>1</MsgID>
    <Target>
      <LocURI>http://www.syncml.org/mgmt-server</LocURI>
    </Target>
    <Source>
      <LocURI>IMEI:493005100592800</LocURI>
    </Source>
    <Cred> <!-- Client credentials are mandatory if the transport layer is
    not providing authentication.-->
      <Meta>
        <Type xmlns="syncml:metinf">syncml:auth-basic</Type>
      </Meta>
      <Data>
        <!—base64 formatting of userid:password -->
      </Data>
    </Cred>
    <Meta> <!-- Maximum message size for the client -->
      <MaxMsgSize xmlns="syncml:metinf">5000</MaxMsgSize>
    </Meta>
  </SyncHdr>
  <SyncBody>
    <Alert>
      <CmdID>1</CmdID>
      <Data>1200</Data> <!-- Server-initiated session -->
    </Alert>
    <Replace>
      <CmdID>3</CmdID>
      <Item>
        <Source><LocURI>./DevInfo/DevID</LocURI></Source>
        <Meta>
          <Format xmlns='syncml:metinf'>chr</Format>
          <Type xmlns='syncml:metinf'>text/plain</Type>
        </Meta>
        <Data>493005100592800</Data>
      </Item>
      <Item>
        <Source><LocURI>./DevInfo/Man</LocURI></Source>
        <Meta>
          <Format xmlns='syncml:metinf'>chr</Format>
          <Type xmlns='syncml:metinf'>text/plain</Type>
          </Meta>
          <Data>Device Factory, Inc.</Data>
        </Item>
        <Item>
          <Source><LocURI>./DevInfo/Mod</LocURI></Source>
          <Meta>
            <Format xmlns='syncml:metinf'>chr</Format>
            <Type xmlns='syncml:metinf'>text/plain</Type>
          </Meta>
          <Data>SmartPhone2000</Data>
        </Item>
        <Item>
          <Source><LocURI>./DevInfo/Mod</LocURI></Source>
          <Meta>
            <Format xmlns='syncml:metinf'>chr</Format>
            <Type xmlns='syncml:metinf'>text/plain</Type>
          </Meta>
          <Data>Smartie</Data>
        </Item>
        <Item>
          <Source><LocURI>./DevInfo/DmV</LocURI></Source>
          <Meta>
            <Format xmlns='syncml:metinf'>chr</Format>
            <Type xmlns='syncml:metinf'>text/plain</Type>
          </Meta>
          <Data>1.0.0.1</Data>
        </Item>
        <Item>
          <Source><LocURI>./DevInfo/Lang</LocURI></Source>
          <Meta>
            <Format xmlns='syncml:metinf'>chr</Format>
            <Type xmlns='syncml:metinf'>text/plain</Type>
          </Meta>
          <Data>US-en</Data>
        </Item>
      </Replace>
      <Final/>
    </SyncBody>
</SyncML>
```

14.1.2 Package 2: Initialization from server to client

```
<SyncML xmlns='SYNCML:SYNCML1.1'>
  <SyncHdr>
    <VerDTD>1.1</VerDTD>
    <VerProto>DM/1.1</VerProto>
    <SessionID>1</SessionID>
    <MsgID>1</MsgID>
    <Source>
      <LocURI>http://www.syncml.org/mgmt-server</LocURI>
    </Source>
    <Target>
      <LocURI>IMEI:493005100592800</LocURI>
    </Target>
    <Cred> <!—Server credentials -->
      <Meta>
        <Type xmlns="syncml:metinf">syncml:auth-basic</Type>
        </Meta>
        <Data><!-- base64 formatting of userid:password --></Data>
      </Cred>
   </SyncHdr>
   <SyncBody>
      <Status>
         <MsgRef>1</MsgRef><CmdRef>0</CmdRef>
         <Cmd>SyncHdr</Cmd>
         <CmdID>6</CmdID>
         <TargetRef>http://www.syncml.org/mgmt-server</TargetRef>
         <SourceRef>IMEI: 493005100592800</SourceRef>
         <!—Authenticated for the session -->
         <Data>212</Data>
      </Status>
      <Status>
         <MsgRef>1</MsgRef><CmdRef>1</CmdRef>
         <CmdID>7</CmdID>
         <Cmd>Alert</Cmd>
         <Data>200</Data><!—OK -->
      </Status>
      <Status>
         <MsgRef>1</MsgRef><CmdRef>3</CmdRef>
         <CmdID>8</CmdID>
         <Cmd>Replace</Cmd>
         <SourceRef>./devinf10</SourceRef>
         <Data>200</Data><!-- OK -->
      </Status>
      <Sequence>
         <CmdID>1</CmdID>
         <Alert>
            <CmdID>2</CmdID>
            <Data>1101</Data> <!-- User confirmation required -->
            <Item></Item>
            <Item>
              <Data>Do you want to add the CNN access point?</Data>
            </Item>
         </Alert>
         <Replace>
            <CmdID>4</CmdID>
            <Meta>
               <Format xmlns="syncml:metinf">b64</Format>
               <Type xmlns="syncml:metinf">
                  application/vnd.wap.connectivity-wbxml
               </Type>
            </Meta>
            <Item>
               <!-- CNN WAP settings object in the settings -->
               <Target>
                  <LocURI>./settings/wap_settings/CNN</LocURI>
               </Target>
               <Data><!-- Base64-coded WAP connectivity document --></Data>
            </Item>
         </Replace>
      </Sequence>
      <Final/>
   </SyncBody>
</SyncML>
```

14.1.3 Package 3: Client response

```
<SyncML xmlns='SYNCML:SYNCML1.1'>
  <SyncHdr>
    <VerDTD>1.1</VerDTD>
    <VerProto>DM/1.1</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target>
      <LocURI>http://www.syncml.org/mgmt-server</LocURI>
    </Target>
    <Source>
      <LocURI>IMEI:493005100592800</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <MsgRef>1</MsgRef>
      <CmdID>1</CmdID>
      <CmdRef>0</CmdRef>
      <Cmd>SyncHdr</Cmd>
      <!—SyncHdr accepted -->
      <Data>212</Data>
    </Status>
    <Status>
      <MsgRef>1</MsgRef>
      <CmdID>2</CmdID>
      <CmdRef>1</CmdRef>
      <Cmd>Sequence</Cmd>
      <!—Sequence executed correctly -->
      <Data>200</Data>
    </Status>
    <Status>
      <MsgRef>1</MsgRef><CmdRef>2</CmdRef>
      <CmdID>3</CmdID>
      <Cmd>Alert</Cmd>
      <!—OK, the user confirmed the action-->
      <Data>200</Data>
    </Status>
    <Status>
      <MsgRef>1</MsgRef>
      <CmdRef>4</CmdRef>
      <CmdID>4</CmdID>
      <Cmd>Replace</Cmd>
      <TargetRef>./settings/wap_settings/CNN</TargetRef>
      <!—OK, access point added-->
      <Data>200</Data>
    </Status>
    <Final/>
  </SyncBody>
</SyncML>
```

14.1.4 Package 4: Acknowledgement of client status

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

```
<SyncML xmlns='SYNCML:SYNCML1.1'>
   <SyncHdr>
      <VerDTD>1.1</VerDTD>
      <VerProto>DM/1.1</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
      <Source>
         <LocURI>http://www.syncml.org/mgmt-server</LocURI>
      </Source>
      <Target>
         <LocURI>IMEI:493005100592800</LocURI>
      </Target>
   </SyncHdr>
   <SyncBody>
      <Status>
         <MsgRef>2</MsgRef>
         <CmdID>1</CmdID>
         <CmdRef>0</CmdRef>
         <Cmd>SyncHdr</Cmd>
         <Data>200</Data>
      </Status>
      <Final/>
   </SyncBody>
</SyncML>
```

14.2 Two-step protocol initiated by the server

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

14.2.1 Package 1: Initialization from client to server

```
<SyncML xmlns='SYNCML:SYNCML1.1'>
   <SyncHdr>
      <VerDTD>1.1</VerDTD>
      <VerProto>DM/1.1</VerProto>
      <SessionID>1</SessionID>
      <MsgID>1</MsgID>
      <Target>
         <LocURI>http://www.syncml.org/mgmt-server</LocURI>
      </Target>
      <Source>
         <LocURI>IMEI:493005100592800</LocURI>
      </Source>
      <Cred> <!—Client credentials are optional -->
         <Meta>
            <Type xmlns="syncml:metinf">syncml:auth-basic</Type>
         </Meta>
         <Data><!-- base64 formatting of userid:password --></Data>
      </Cred>
```

```
     <Meta><!-- Maximum message size for the client -->
       <MaxMsgSize xmlns="syncml:metinf">5000</MaxMsgSize>
     </Meta>
   </SyncHdr>
   <SyncBody>
     <Alert>
       <CmdID>1</CmdID>
       <Data>1200</Data> <!-- Server-initiated session -->
     </Alert>
     <Replace>
       <CmdID>3</CmdID>
       <Item>
         <Source><LocURI>./DevInfo/DevID</LocURI></Source>
         <Meta>
           <Format xmlns='syncml:metinf'>chr</Format>
           <Type xmlns='syncml:metinf'>text/plain</Type>
         </Meta>
         <Data>493005100592800</Data>
       </Item>
       <Item>
         <Source><LocURI>./DevInfo/Man</LocURI></Source>
         <Meta>
           <Format xmlns='syncml:metinf'>chr</Format>
           <Type xmlns='syncml:metinf'>text/plain</Type>
         </Meta>
         <Data>Device Factory, Inc.</Data>
       </Item>
       <Item>
         <Source><LocURI>./DevInfo/Mod</LocURI></Source>
         <Meta>
           <Format xmlns='syncml:metinf'>chr</Format>
           <Type xmlns='syncml:metinf'>text/plain</Type>
         </Meta>
         <Data>SmartPhone2000</Data>
       </Item>
       <Item>
         <Source><LocURI>./DevInfo/Mod</LocURI></Source>
         <Meta>
           <Format xmlns='syncml:metinf'>chr</Format>
           <Type xmlns='syncml:metinf'>text/plain</Type>
         </Meta>
         <Data>Smartie</Data>
       </Item>
       <Item>
         <Source><LocURI>./DevInfo/DmV</LocURI></Source>
         <Meta>
           <Format xmlns='syncml:metinf'>chr</Format>
           <Type xmlns='syncml:metinf'>text/plain</Type>
         </Meta>
         <Data>1.0.0.1</Data>
       </Item>
       <Item>
         <Source><LocURI>./DevInfo/Lang</LocURI></Source>
         <Meta>
           <Format xmlns='syncml:metinf'>chr</Format>
           <Type xmlns='syncml:metinf'>text/plain</Type>
         </Meta>
         <Data>US-en</Data>
       </Item>
```

```
      </Replace>
      <Final/>
   </SyncBody>
</SyncML>
```

14.2.2 Package 2: Initialization from server to client

```
<SyncML xmlns='SYNCML:SYNCML1.1'>
   <SyncHdr>
      <VerDTD>1.1</VerDTD>
      <VerProto>DM/1.1</VerProto>
      <SessionID>1</SessionID>
      <MsgID>1</MsgID>
      <Source>
         <LocURI>http://www.syncml.org/mgmt-server</LocURI>
      </Source>
      <Target>
         <LocURI>IMEI:493005100592800</LocURI>
      </Target>
      <Cred> <!—Server credentials -->
         <Meta>
            <Type xmlns="syncml:metinf">syncml:auth-basic</Type>
         </Meta>
         <Data><!-- base64 formatting of userid:password --></Data>
      </Cred>
   </SyncHdr>
   <SyncBody>
      <Status>
         <MsgRef>1</MsgRef><CmdRef>0</CmdRef>
         <Cmd>SyncHdr</Cmd>
         <CmdID>5</CmdID>
         <TargetRef>http://www.syncml.org/mgmt-server</TargetRef>
         <SourceRef>IMEI: 493005100592800</SourceRef>
         <!—Authenticated for the session -->
         <Data>212</Data>
      </Status>
      <Status>
         <MsgRef>1</MsgRef><CmdRef>1</CmdRef>
         <CmdID>6</CmdID>
         <Cmd>Alert</Cmd>
         <Data>200</Data><!—OK -->
      </Status>
      <Status>
         <MsgRef>1</MsgRef><CmdRef>3</CmdRef>
         <CmdID>7</CmdID>
         <Cmd>Replace</Cmd>
         <SourceRef>./devinf10</SourceRef>
         <Data>200</Data><!-- OK -->
      </Status>
      <Alert>
         <CmdID>2</CmdID>
         <Data>1100</Data> <!—- User displayable notification -->
         <Item></Item>
         <Item>
            <Data>Your antivirus software is being updated</Data>
         </Item>
      </Alert>
```

```
     <!-- Let's get the installed antivirus definition version number now -
->
     <Get>
       <CmdID>4</CmdID>
       <Item>
         <Target>
          <LocURI>./antivirus_data/version</LocURI>
         </Target>
       </Item>
     </Get>
     <Final/>
   </SyncBody>
</SyncML>
```

14.2.3 Package 3: Client response

```
<SyncML xmlns='SYNCML:SYNCML1.1'>
   <SyncHdr>
     <VerDTD>1.1</VerDTD>
     <VerProto>DM/1.1</VerProto>
     <SessionID>1</SessionID>
     <MsgID>2</MsgID>
     <Target>
       <LocURI>http://www.syncml.org/mgmt-server</LocURI>
     </Target>
     <Source>
       <LocURI>IMEI:493005100592800</LocURI>
     </Source>
   </SyncHdr>
   <SyncBody>
     <Status>
       <MsgRef>1</MsgRef>
       <CmdID>1</CmdID>
       <Cmd>SyncHdr</Cmd>
       <Data>212</Data>
     </Status>
     <Status>
       <MsgRef>1</MsgRef>
       <CmdRef>2</CmdRef>
       <CmdID>2</CmdID>
       <Cmd>Alert</Cmd>
       <Data>200</Data><!—User notification OK -->
     </Status>
     <Status>
       <MsgRef>1</MsgRef>
       <CmdRef>4</CmdRef>
       <CmdID>2</CmdID>
       <Cmd>Get</Cmd>
       <TargetRef>./antivirus_data/version</TargetRef>
       <Data>200</Data><!-- Get OK -->
     </Status>
     <!-- Results for the Get: antivirus version number -->
     <Results>
       <MsgRef>1</MsgRef><CmdRef>4</CmdRef>
       <CmdID>3</CmdID>
       <Item>
```

```
          <Source>
           <LocURI>./antivirus_data/version</LocURI>
          </Source>
          <Data>antivirus-inc/20010522b/5</Data>
        </Item>
      </Results>
      <Final/>
   </SyncBody>
</SyncML>
```

14.2.4 Package 4: Continue with management operations

```
<SyncML xmlns='SYNCML:SYNCML1.1'>
   <SyncHdr>
      <VerDTD>1.1</VerDTD>
      <VerProto>DM/1.1</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
      <Target>
         <LocURI>http://www.syncml.org/mgmt-server</LocURI>
      </Target>
      <Source>
         <LocURI>IMEI:493005100592800</LocURI>
      </Source>
   </SyncHdr>
   <SyncBody>
      <Status>
         <MsgRef>2</MsgRef>
         <CmdID>1</CmdID>
         <Cmd>SyncHdr</Cmd>
         <Data>212</Data>
      </Status>
      <!-- Send now antivirus updates -->
      <Replace>
         <CmdID>2</CmdID>
         <Meta>
            <Format xmlns="syncml:metinf">b64</Format>
            <Type xmlns="syncml:metinf">
               application/antivirus-inc.virusdef
            </Type>
         </Meta>
         <Item>
            <Target>
             <LocURI>./antivirus_data</LocURI>
            </Target>
            <Data><!-- Base64-coded antivirus file --></Data>
         </Item>
      </Replace>
      <Final/>
   </SyncBody>
</SyncML>
```

14.2.5 Restart protocol iteration with Package 3: Client response

```
<SyncML xmlns='SYNCML:SYNCML1.1'>
```

```
  <SyncHdr>
    <VerDTD>1.1</VerDTD>
    <VerProto>DM/1.1</VerProto>
    <SessionID>1</SessionID>
    <MsgID>3</MsgID>
    <Target>
      <LocURI>http://www.syncml.org/mgmt-server</LocURI>
    </Target>
    <Source>
      <LocURI>IMEI:493005100592800</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <MsgRef>2</MsgRef>
      <CmdID>1</CmdID>
      <Cmd>SyncHdr</Cmd>
      <Data>200</Data>
    </Status>
    <Status>
      <MsgRef>1</MsgRef>
      <CmdRef>2</CmdRef>
      <CmdID>2</CmdID>
      <Cmd>Replace</Cmd>
      <TargetRef>./antivirus_data</TargetRef>
      <!-- OK, antivirus update loaded-->
      <Data>200</Data>
    </Status>
    <Final/>
  </SyncBody>
</SyncML>
```

14.2.6 Package 4: Finish the protocol, no continuation

```
<SyncML xmlns='SYNCML:SYNCML1.1'>
  <SyncHdr>
    <VerDTD>1.1</VerDTD>
    <VerProto>DM/1.1</VerProto>
    <SessionID>1</SessionID>
    <MsgID>3</MsgID>
    <Source>
      <LocURI>http://www.syncml.org/mgmt-server</LocURI>
    </Source>
    <Target>
      <LocURI>IMEI:493005100592800</LocURI>
    </Target>
  </SyncHdr>
  <SyncBody>
    <Status>
      <MsgRef>3</MsgRef>
      <CmdID>1</CmdID>
      <CmdRef>0</CmdRef>
      <Cmd>SyncHdr</Cmd>
      <Data>200</Data>
    </Status>
    <Final/>
```

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