Email data object specification

Approved Version 1.2.1 – 10 Aug 2007

Open Mobile Alliance

OMA-TS-DS_DataObjEmail-V1_2_1-20070810-A

Contents

Tables

1. Scope

The **email** data object is presented in this document. The content-specific aspects of synchronization (filtering keywords, etc…) are listed and clarified.

2. References

2.1 Normative References

[DSDEVDTD] "OMA DS Device Information DTD", Open Mobile Alliance™," OMA-SUP-DS-DevInf-DTD-V1_2,

URL:http://www.openmobilealliance.org

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

[ISO 8601] "Data elements and interchange formats- Information interchange--Representation of dates and times ISO 8601-2000", International Organization for Standardization, June, 1988, URL://www.iso.ch/iso/en/ISOOnline.openerpage

[RFC1327] Hardcastle-Kille, S., "Mapping between X.400 (1988) / ISO 10021 and RFC 822", RFC 1327, May 1992. URL:http://www.ietf.org/rfc/rfc1327.txt

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

[RFC2046] "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", N. Freed & N. Borenstein, November 1996, URL:http://www.ietf.org/rfc/rfc2046.txt

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

[RFC2234] "Augmented BNF for Syntax Specifications: ABNF", D. Crocker, Ed., P. Overell,

November 1997, URL:http://www.ietf.org/rfc/rfc2234.txt

[RFC2822] "Internet Message Format". P. Resnick, Editor. April 2001 URL:http://www.ietf.org/rfc/rfc2822.txt

[WBXML] "WAP Binary XML Content Format Specification." WAP Forum.

URL:http://www1.wapforum.org/tech/terms.asp?doc=WAP-192-WBXML-20010725-a.pdf

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

2.2 Informative References

none

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

3.2 Definitions

Data type	The schema used to represent a data object (e.g., text/calendar MIME content type for an iCalendar representation of calendar information or text/directory MIME content type for a vCard representation of contact information).

3.3 Abbreviations

OMA	Open Mobile Alliance

4. Introduction

The email data object can be used to represent an interpersonal electronic mail object.

5. XML Usage

The **email** objects are represented in a mark-up language defined by [XML]. The **email** is an XML application. The **email** DTD (Document Type Definition) defines the XML document type used to represent **email** object. The **mail** DTD can be found in Section 9, but it is not necessary to read the DTD in order to understand it.

Email objects are specified using well-formed XML. However, the **email** need not be valid XML. That is, the **email** objects do not need to specify the XML declaration or prolog. They only need to specify the body of the XML document. This restriction allows for the **email** objects to be specified with greater terseness than well-formed, valid XML documents.

5.1 Including email object in other XML documents

When using the **email** object as content of another XML document element the mark-up characters MUST be properly escaped or the CDATA sections MUST be used. See [XML] specification for details on character escaping and usage of CDATA sections. See section 10.3 of this document for an example.

5.2 XML Namespaces

Email objects to date have no elements that may include elements of the other namespaces.

5.3 XML Attributes

In order to simplify the implementation of the **email** on small devices, the **email** objects have been intentionally designed to use the XML elements only. Currently no XML attributes are being defined for **email** objects.

5.4 WBXML

XML can be viewed as more verbose than alternative binary representations. This is often cited as a reason why it may not be appropriate for low bandwidth network protocols. In most cases, **email** uses shortened element type. This provides a minor reduction in verbosity.

Additionally, the **email** objects can be encoded in a tokenized, binary format defined by [WBXML]. The use of [WBXML] format is external to specification of the **email** and should be transparent to any application. The combination of the use of shortened element type names and an alternative binary format makes **email** competitive, from a compressed format perspective, with alternative, but private, binary representations.

6. MIME Usage

The [RFC2045] Internet standard provides an industry-accepted mechanism for identifying different content types. The **email** object is identified by a MIME media type. The **application/vnd.omads-email+xml** MIME content type MUST be used to indicate the **email** object wherever such indication is required.

7. Data types

The following basic data type definitions are provided for referencing from other parts of this document.

7.1 datetime

Usage: This value type is used to identify values that specify a precise calendar date and time of day.

Description:

The *datetime* data type is used to identify values that contain a precise calendar date and time of day. The format is based on the [ISO 8601] complete representation, basic format for a calendar date and time of day. The text format is a concatenation of the "date", followed by the LATIN CAPITAL LETTER T character (US-ASCII decimal 84) time designator, followed by the "time" format.

The *datetime* data type expresses time values in two forms:

The form of date and time with UTC offset MUST NOT be used. For example, the following is not valid for a date-time value:

```
...
<datefield>19980119T230000-0800</datefield> <!-- Invalid time format -->
...
```

FORM #1: DATE WITH LOCAL TIME

The date with local time form is simply a date-time value that does not contain the UTC designator nor does it reference a time zone. For example, the following represents January 18, 1998, at 11 PM:

```
...
<datefield>19980118T230000</datefield> <!-- January 18, 1998, 11 PM -->
...
```

This notation of *datetime* type is to be used by devices that have no knowledge of the time zone in which they operate. In this case, the *datetime* value that is being transferred is usually the same as the value that is being stored and shown to the user in the application UI.

FORM #2: DATE WITH UTC TIME

The date with UTC time, or absolute time, is identified by a LATIN CAPITAL LETTER Z suffix character (US-ASCII decimal 90), the UTC designator, appended to the time value. For example, the following represents January 19, 1998, at 0700 UTC:

```
...
<datefield>19980119T070000Z</datefield> <!-- January 19,1998,07:00 UTC -->
...
```

7.2 bool

Usage: To be used for Boolean type fields

Restrictions: A text value that MUST be either "`true`" to indicate Boolean "`true`" or "`false`" to indicate "`false`". If the field is not present its value is assumed to be "`false`".

Example:

```
...
<booleanfield>true</booleanfield> <!-- the field is set to "true" -->
...
```

7.3 text

Usage: To be used for textual fields

Restrictions: If the field is not present its value is assumed to be an empty string.

Example:

```
...
<textfield>Hello World!</textfield>
...
```

7.4 int

Usage: To be used for integer numeric fields.

Restrictions: The format for the integer values is defined here in an ABNF notation [RFC2234].

```
nonzero-digit = "1"/ "2"/ "3"/ "4"/ "5"/ "6"/ "7"/ "8"/ "9"

octal-digit = "0"/ "1"/ "2"/ "3"/ "4"/ "5"/ "6"/ "7"

hexadecimal-digit = "0"/ "1"/ "2"/ "3"/ "4"/ "5"/ "6"/ "7"/ "8"/ "9"
/                   "a"/ "b" / "c"/ "d"/ "e"/ "f"
/                   "A"/ "B"/ "C"/ "D"/ "E"/ "F"

decimal-constant = nonzero-digit *("0" / nonzero-digit)
hexadecimal-constant = ("0x" / "0X") 1*hexadecimal-digit
```

```
octal-constant = "0" 1*octal-digit


integer-value = *1("+" / "-") (decimal-constant / hexadecimal-constant /
octal-constant)
```

Example:

```
...
<negativevalue>-1234</negativevalue>
<positivevalue>1234</positivevalue>
<anotherpositivevalue>+0xfffabc5</anotherpositivevalue>
<octal>010</octal> <!-- octal value equivalent to decimal 8 -->
...
```

8. Mark-up Language Description

8.1 Email

Usage: Indicates the beginning of the object

Parent elements: None

Content model:

```
Email (read?, forwarded?, replied?, received?, created?, modified?,
deleted?, flagged?, emailitem?, Ext*)
```

8.2 read

Usage: Specifies whether the email has been read

Parent elements: `Email`

Restrictions: *bool* type field as specified in section 7.2.

Content model:

```
read (#PCDATA)
```

8.3 forwarded

Usage: Specifies whether the email has been forwarded

Parent elements: `Email`

Restrictions: *bool* type field as specified in section 7.2.

Content model:

```
forwarded (#PCDATA)
```

8.4 replied

Usage: Specifies whether the email has been replied

Parent elements: `Email`

Restrictions: *bool* type field as specified in section 7.2.

Content model:

```
replied (#PCDATA)
```

8.5 received

Usage: Specifies the date and time when the email was received

Parent elements: `Email`

Restrictions: *datetime* type field as specified in section 7.1.

Content model:

```
received (#PCDATA)
```

8.6 created

Usage: Specifies the date and time when the email was created

Parent elements: `Email`

Restrictions: *datetime* type field as specified in section 7.1.

Content model:

```
created (#PCDATA)
```

8.7 modified

Usage: Specifies the date and time when any of the email object fields was last changed. Initially (after the draft item is created or a new message received) the `modified` date is equal to `created` date. As the state of the object changes (e.g. draft message gets submitted or unread message gets read) the `modified` field gets updated to reflect the date and time of the last state change.

Parent elements: `Email`

Restrictions: *datetime* type field as specified in section 7.1.

Content model:

```
modified (#PCDATA)
```

8.8 deleted

Usage: Specifies whether the email has been scheduled for deletion

Parent elements: `Email`

Restrictions: *bool* type field as specified in section 7.2.

Content model:

```
deleted (#PCDATA)
```

8.9 flagged

Usage: Specifies whether the email has been flagged

Parent elements: `Email`

Restrictions: *bool* type field as specified in section 7.2.

Content model:

```
flagged (#PCDATA)
```

8.10 emailitem

Usage: contains the email header and body as specified in RFC822 / RFC2822

Parent elements: `Email`

Restrictions: The supporters MUST implement this property. If the field is not present within the object, the object is assumed to have an empty body.

Content model:

```
emailitem (#PCDATA)
```

8.10.1 enc

Usage: declares the mechanism used to encode the content of the element. This is used to avoid corrupting the XML content of the element with the presence of characters which do not belong to the valid ranges of characters as defined by the [XML]

Parent elements: `emailitem`

Restrictions: The following table lists standard `enc` values that MUST be understood by the conforming implementations.

Enc	Description
“quoted-printable”	The contents of the element is encoded using quoted-printable algorithm as specified by the section 6.7 of the [RFC2045]
“base64”	The contents of the element is encoded using quoted-printable algorithm as specified by the section 6.8 of the [RFC2045]

Table 1 enc attribute values

If the `enc` attribute is not present, the content is assumed to have no encoding.

The implementations SHOULD NOT use other `enc` attribute values than specified in the [Table 1]. In case of other `enc` values the usage of these encodings MUST conform to the rules defined by the[RFC2045] for `Content-Transfer-Encoding`.

Content model:

```
emailitem enc (CDATA #IMPLIED)
```

8.11 Extension fields

8.11.1 Unique naming

If an extension field is required, the following naming convention MUST be followed in order to prevent undesirable field name collisions.

```
x-name        = "x-" vendorid "-" 1*(ALPHA / DIGIT / "-")          ;field name

vendorid      = 3*(ALPHA / DIGIT)                    ;Vendor identification

ALPHA         = %x41-5A / %x61-7A                    ; A-Z / a-z

DIGIT         = %x30-39                              ; 0-9
```

8.11.2 Ext

Usage: Specifies the non-standard, experimental extensions supported by the device. The extensions are specified in terms of the XML element type name and the value.

Parent Elements: `Email`

Restrictions: The `Ext` element type MUST specify the extension element name. It may also specify one or more enumerated values. Multiple non-standard extensions can be specified by specifying the `Ext` element type multiple times. This element type is optional.

Content Model:

```
Ext (XNam, XVal*)
```

Attributes: None.

Example: The following example specifies a non-standard extension, named "`CliVer`" for a fictitious company, Foo, which takes values of "`5.0`", "`5.01`" or "`5.02`".

```
<Ext>
   <XNam>x-Foo-CliVer</XNam>
   <XVal>5.0</XVal>
   <XVal>5.01</XVal>
   <XVal>5.02</XVal>
</Ext>
```

8.11.3 XNam

Usage: Specifies the name of one of the extension element types.

Parent Elements: `Ext`

Restrictions: The element type is required whenever an Ext element is present.

Content Model:

```
XNam (#PCDATA)
```

Attributes: None.

Example:

```
<Ext>
  <XNam>x-Foo-CliVer</XNam>
  <XVal>5.0</XVal>
  <XVal>5.01</XVal>
  <XVal>5.02</XVal>
</Ext>
```

8.11.4 XVal

Usage: Specifies one of the valid values for an extension element type.

Parent Elements: `Ext`

Restrictions:

Content Model:

```
XVal (#PCDATA)
```

Attributes: None.

Example:

```
<Ext>
  <XNam>x-Foo-CliVer</XNam>
  <XVal>5.0</XVal>
  <XVal>5.01</XVal>
  <XVal>5.02</XVal>
</Ext>
```

9. DTD

```
<!--

application/vnd.omads-email+xml V1.2 Document Type Definition


http://www.openmobilealliance.org/tech/DTD/OMA-DS-DataObjEmail-DTD-
V1_2.dtd


Copyright Open Mobile Alliance Ltd., 2002-2003

          All rights reserved

Terms and conditions of use are available from the

Open Mobile Alliance Ltd. web site at

http://www.openmobilealliance.org/useterms.html-->


<?xml version="1.0" encoding="UTF-8"?>

<!-- Root Element -->

<!ELEMENT Email (read?, forwarded?, replied?, received?, created?,
modified?, deleted?, flagged?, emailitem?, Ext*)>


<!ELEMENT read (#PCDATA)>

<!ELEMENT forwarded (#PCDATA)>

<!ELEMENT replied (#PCDATA)>

<!ELEMENT received (#PCDATA)>

<!ELEMENT created (#PCDATA)>

<!ELEMENT modified (#PCDATA)>

<!ELEMENT deleted (#PCDATA)>

<!ELEMENT flagged (#PCDATA)>

<!ELEMENT emailitem (#PCDATA)>

<!ATTLIST emailitem enc CDATA #IMPLIED>

<!ELEMENT Ext (XNam, XVal*)>

<!ELEMENT XNam (#PCDATA)>

<!ELEMENT XVal (#PCDATA)>

<!-- End of DTD Definition -->
```

10.SyncML Data Synchronization Usage

The following sections describe the content-specific recommendations for using the data synchronization [DSREPU] protocol with **email** data objects.

10.1 CTCap

Refer to [DSDEVDTD] for further details on the specification of the Device Information DTD.

```
<CTCap>
   <CTType> application/vnd.omads-email+xml</CTType>
   <Property>
      <PropName>read</PropName>
      <DataType>bool</DataType>
      <DisplayName>Read</DisplayName>
   </Property>
   <Property>
      <PropName>forwarded</PropName>
      <DataType>bool</DataType>
      <DisplayName>Forwarded</DisplayName>
   </Property>
   <Property>
      <PropName>replied</PropName>
      <DataType>bool</DataType>
      <DisplayName>Replied</DisplayName>
   </Property>
   <Property>
      <PropName>received</PropName>
      <DataType>datetime</DataType>
      <DisplayName>Date received</DisplayName>
   </Property>
   <Property>
      <PropName>created</PropName>
      <DataType>datetime</DataType>
      <DisplayName>Date created</DisplayName>
```

```
   </Property>
   <Property>
      <PropName>emailitem</PropName>
      <DataType>bin</DataType>
      <DisplayName>emailitem</DisplayName>
      <PropParam>
            <ParamName>enc</ParamName>
            <ValEnum>base64</ValEnum>
            <ValEnum>quoted-printable</ValEnum>
      </PropParam>
      <PropParam>
            <ParamName>texttype</ParamName>
            <ValEnum>text/plain</ValEnum>
            <ValEnum>text/html</ValEnum>
      </PropParam>
      <PropParam>
            <ParamName>attachtype</ParamName>
            <ValEnum>image/jpeg</ValEnum>
            <ValEnum>image/tiff</ValEnum>
      </PropParam>
   </Property>
</CTCap>
```

10.1.1 enc

The `PropParam` element type with the value of `enc` MUST utilise `ValEnum` element type to indicate the supported encoding algorithms. The example above (10.1) illustrates a section of `CTCap` that lists the standard encoding algorithms as supported.

10.1.2 texttype

This `PropParam` specifies which content types are recognized for textual parts of the message. If no enumeration values are specified, all content types are allowed. If `texttype` is not present within the `CTCap`, the sender does not require textual information. This particular case is useful for scenarios in which the sender chooses to synchronize only limited information for the messages (e.g. [RFC2822] headers only).

Each enumeration value can contain one of the following optional prefix:

- "filter-out:" means that any message textual parts with this content type are not required by the sender. It is allowed to have several enumeration values with the prefix "filter-out:".

- "transcode:" means that the sender would like to receive all the textual parts of the sent messages transcoded to this content type. It is allowed to have several enumeration values with the prefix "transcode:".

10.1.3 attachtype

This `PropParam` specifies which content types are recognized for attached objects. If no enumeration values are specified, all content types are allowed. If attachtype is not present within the `CTCap`, attached objects are not required by the sender. This particular case is useful for scenarios in which the sender chooses to synchronize only limited information for the messages (e.g. [RFC2822] headers only).

Each enumeration value can contain one of the following optional prefix:

- "filter-out:" means that any attached objects with this content type are not required by the sender. It is allowed to have several enumeration values with the prefix "filter-out:".
- "transcode:" means that the sender would like to receive any attached objects transcoded to this content type. It is allowed to have several enumeration values with the prefix "transcode:".

10.2 Data Sync Record and Field Level Filtering

10.2.1 Email Media Object Filter

Filtering for email objects can be specified using both `Record` and `Field` elements.

In the case of `Record` elements, the set of recommended keywords to support are as follows:

```
ct-filter-keyword = email-field | search-keyword
```

10.2.1.1 email-field

```
email-field = <Any field that is defined for the application/vnd.omads-
email+xml content type in this document except for the emailitem field>
```

10.2.1.2 search-keyword

In addition to the actual email fields defined in this document, the following set of keywords is recommended to be supported so that they can be specified in a filtering query to limit the amount of items. The types of values to be used with these keywords are described in section 7.

Keyword	Interpretation	ct-filter-value Type
BCC	Contents of the [RFC2822] "`Bcc:`" destination address field	text
CC	Contents of the [RFC2822] "`Cc:`" destination address field	text
FROM	Contents of the [RFC2822] "`From:`" originator field	text
IMPORTANCE	The case-insensitive values "`low`", "`normal`" and "`high`" are allowed as specified by the [RFC1327].	text

NOATTACH	Item does not contain attachments	bool
NOBODY	Item does not contain [RFC2822] body (i.e. item is an empty message with only [RFC2822] header fields defined)	bool
SIZE	The number of octets in the content of `emailitem` field	int
SUBJECT	Contents of the [RFC2822] "`Subject:`" informational field	text
TO	Contents of the [RFC2822] "`To:`" destination address field	text

Table 2 search keywords

10.2.1.3 Example 1: Record filtering

In this scenario, the client wishes to synchronize e-mail messages whose internal date is within or later than June 1, 2003 and importance is either "`normal`" or "`high`"

1. During the initial sync, the client and server exchange their device info.
2. The client analyses the server's device info, and the client notes that the server supports receiving filters on the Email data store for queries using the "`syncml:filtertype-cgi`" grammar.
    a. The server includes in its device info the Filter-Rx and FilterCap elements that it supports. This includes the "`IMPORTANCE`" and "`created`" keywords.
    b. The client doesn't require filtering on any additional fields, so it determines that this server supports the filter it wishes to send.

```
<Datastore>
  <SourceRef>./email/inbox</SourceRef>
  <DisplayName>Email Inbox</DisplayName>
  ...
  <Filter-Rx>
    <CTType>syncml:filtertype-cgi</CTType>
    <VerCT>1.0</VerCT>
  </Filter-Rx>
  <CTCap>
    ...
  </CTCap>
  <FilterCap>
    <CTType>syncml:filtertype-cgi</CTType>
    <VerCT>1.0</VerCT>
    <FilterKeyword>IMPORTANCE</FilterKeyword>
    <FilterKeyword>created</FilterKeyword>
  </FilterCap>
</Datastore>
```

3. The client sends an `Alert` for the Email data store with a filter.

    a. It includes a `Filter Record` element with a Meta Type value of "`syncml:filtertype-cgi`" to indicate the grammar being used.

    b. The filter query in the `Item Data` element contains a value of "`created&GT;20030601T000000Z&AND;IMPORTANCE&NE;low`" to constrain the items synchronized to those whose internal date is within or later than June 1, 2003 and importance is either "`normal`" or "`high`".

```
<Alert>
  <Data>200</Data>
  <Item>
    <Target>
      <LocURI>./email/inbox</LocURI>
      <Filter>
        <Meta><Type>application/vnd.omads-email+xml</Type></Meta>
        <Record>
          <Item>
            <Meta><Type>syncml:filtertype-cgi</Type></Meta>
            <Data>
              created&GT;20030601T000000Z&AND;IMPORTANCE&NE;low
            </Data>
          </Item>
        </Record>
      </Filter>
    </Target>
    <Source>
      <LocURI>dev-inbox</LocURI>
    </Source>
  </Item>
</Alert>
```

4. The server receives the `Alert` with the `Filter` `Record` element.
    a. It determines that it supports the filter operation for the data store, content type, filter grammar, and properties.
    b. It replies with a status code of 200 for the `Alert`, indicating that it can satisfy the request to sync with filtering.
5. The synchronization process continues normally.
    a. The client sends all of its changes to the server (the filter constraint is not imposed on it in this scenario).
    b. The server sends changes only for items that satisfy the filter query.

10.2.1.4 Example 2: Content filtering

In this scenario, the client wishes to synchronize only plain text bodies with all attachments removed and truncate the resulting emailitem content to 2 kilobytes.

1. During the initial sync, the client and server exchange their device info.
2. The client sends an `Alert` for the Email data store with a filter. It includes a `Filter Field` element containing a Property element set to "`emailitem`" containing a MaxSize element set to 2048 (2K), specifying that only "`text/plain`" type parts of the text can be included.

```
<Alert>
  <Data>200</Data>
  <Item>
    <Target>
      <LocURI>./email/inbox</LocURI>
      <Filter>
        <Meta><Type>application/vnd.omads-email+xml</Type></Meta>
        <Field>
          <Item>
            <Meta><Type>application/vnd.syncml-devinf+xml</Type></Meta>
            <Data>
              <Property>
                <PropName>emailitem</PropName>
                <MaxSize>2048</MaxSize>
                <PropParam>
                  <ParamName>texttype</ParamName>
                  <ValEnum>text/plain</ValEnum>
                </PropParam>
              </Property>
            </Data>
          </Item>
        </Field>
      </Filter>
    </Target>
    <Source>
      <LocURI>dev-inbox</LocURI>
    </Source>
  </Item>
</Alert>
```

3. The server receives the `Alert` with the `Filter` `Record` element.

    c. It determines that it supports the filter operation for the data store, content type, filter grammar, and properties.

d. It replies with a status code of 200 for the `Alert`, indicating that it can satisfy the request to sync with filtering.

4. The synchronization process continues normally.

e. The client sends all of its changes to the server (the filter constraint is not imposed on it in this scenario).

f. Server recalculates the `emailitem` fields for the client to include only plain text. For the items that contain more than 2048 bytes in resulting `emailitem` field, the server truncates the field content.

10.3 Email object replace example

```
...
<Sync>
...
   <Replace>
      <CmdID>6</CmdID>
      <Meta>
         <Type xmlns='syncml:metinf'>application/vnd.omads-
email+xml</Type>
      </Meta>
      <Item>
         <Source>
            <LocURI>123</LocURI>
         </Source>
         ...
         <Data><![CDATA[
            <Email>
               <read>false</read>
               <created>20030807T231830Z</created>
               <emailitem enc="base64">
<--!The B64 encoded content of the email goes here -->
               ...

               </emailitem>
            </Email>]]>
         </Data>
      </Item>
```

```
   </Replace>
</Sync>
```

In this example the device that has previously indicated that it supports "created", "read" and "emailitem" fields for the email object data type receives the update of the object.

11. Advanced Email Use Cases Usage

The following sub sections describe some advanced email use cases usage and recommendations with the **email** data object. This section does not represent all the possible email use cases.

11.1 Sending Only the Headers (including attached objects)

In most of the email usage with OMA DS, it is not recommended to transfer the full messages but only the headers, and then the client may request or point to a specific message or message attachment. E.g. requesting a specific message attachment or forwarding a message with attachments but without having to get the attachments on the client side (see the sections below for more details on such use cases). That is, the server must be able to send not only message headers but also attachment headers with a unique identifier that could be used in later commands by the client.

The [RFC2045] proposes the external-body subtype, with the associated MIME type "message/external-body", to indicate that the actual body data is not included (see [RFC2045] section 5.2.3 for more details). Each attached object will need to be specified by two headers: the first one describes how to access the external data, while the nested second one is the header of the referenced data. Note that the Content-ID header field is mandatory to give a unique identifier by which to reference the data. That is, the server will use such mechanism to specify the attachments' header without the need to send the full attachment data.

Example: The following example illustrates an email message content, i.e. placed within the Data element, that only specifies the message headers including both the body and the attachment ones. This email will be used in the following examples to demonstrate Getting the full email and specific attached objects, Smart forward and reply. It is assumed that the email has been synchronized to the client and assigned a LUID = 1234.

Example a. Is the full email including the entire message body **as it would be displayed to the end user (without MIME headers)**. This is for reference for the following examples.

```
From: John Doe <john.doe@server.com>
To: Jane Doe <jane.doe@server.com>
Date: Mon, 9 Apr 2007 08:15:39 -0800 (PST)
Subject: Example of email with attachment

Hi Jane :
Here is a picture of the OMA DS WG at a local restaurant.

OMADSWGimage.jpg
```

Example b. Is the same email with body and attachment truncated and represented by only the MIME headers as it would be sent in an OMA DS message. This is the email representation that is assumed to have been synchronized down to the client and assigned a LUID of 1234. The assumption is that in order to receive this representation of the email, the end user had set filters which truncated the email so that all he received was the headers.

```
Message-Id: <2007040900069@mail.server.com>
From: John Doe <john.doe@server.com>
To: Jane Doe <jane.doe@server.com>
Date: Mon, 9 Apr 2007 08:15:39 -0800 (PST)
Subject: Example of email with attachment
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=some_boundary

--some_boundary
Content-Type: text/plain; charset=us-ascii
```

```
Content-ID: <body_id@mail.server.com>


--some_boundary
Content-Type: message/external-body;
              access-type=mail-server;
              server=mail.server.com
              name="OMADSWGimage.jpg"

Content-Type: image/jpeg
Content-ID: <attach_id@mail.server.com>
Content-Transfer-Encoding: binary
Content-Length: 204800
```

After receiving such email message, the client will be able to request the full email message or only the message body or only a specific attached object.

11.2 Getting a Specific Email Message

Once a client has received a message with partial content, e.g. only the message headers, the Get command is used to retrieve the full message record by sending the record LUID within the Target LocURI (see [RFC2045] section 6.5.7).

Example 1: The following example illustrates how a client can retrieve a full email message that has already been synchronized; but only the email message headers have been synchronized. In this example the email message record LUID is 1234. If the email message has attached objects, then they will also be returned within the result of this Get command.

```
<Get>
  <CmdId>5</CmdId>
  <Meta>
    <Type xmlns='syncml:metinf'>message/rfc2822</Type>
  </Meta>
  <Item>
    <Target>
      <LocURI>emailstore/1234</LocURI>
    </Target>
  </Item>
</Get>
```

Example 2a: The following example, a variant of example 1, illustrates how a client can retrieve a full email message body without attachments this time. The solution here, is to add a filter within the Target element to filter-out all attachments for that specific email message.

```
<Get>
  <CmdId>5</CmdId>
  <Meta>
    <Type xmlns='syncml:metinf'>message/rfc2822</Type>
  </Meta>
  <Item>
    <Target>
      <LocURI>emailstore/1234</LocURI>
      <Filter>
        <Meta><Type>application/vnd.omads-email</Type></Meta>
```

```
        <Field>
          <Item>
            <Meta><Type>application/vnd.syncml-devinf+xml</Type></Meta>
            <Data>
              <Property>
                <PropName>emailitem</PropName>
                <PropParam>
                  <ParamName>texttype</ParamName>
                </PropParam>
              </Property>
            </Data>
          </Item>
        </Field>
      </Filter>
    </Target>
  </Item>
</Get>
```

Example 2b: The following example illustrates an alternative method for a client to retrieve a full email message body without attachments . This example uses the body-id in the GET command. In this example the email message record LUID is 1234, and the Content-ID of the body object is body_id@mail.server.com, as illustrated in the base example above..

```
<Get>
  <CmdId>5</CmdId>
  <Meta>
    <Type xmlns='syncml:metinf'>message/rfc2822</Type>
  </Meta>
  <Item>
    <Target>
      <LocURI>emailstore/1234/body_id@mail.server.com</LocURI>
    </Target>
  </Item>
</Get>
```

Example 3: The following example, a variant of example 1, illustrates how a client can retrieve a full email message body and the attachments transcoded to a specific content type. This example adds a filter within the Target element to specify the transcoding of the attached objects for that specific email message. In this example, all the attached objects will be transcoded to image/png.

```
<Get>
  <CmdId>5</CmdId>
  <Meta>
    <Type xmlns='syncml:metinf'>message/rfc2822</Type>
  </Meta>
  <Item>
    <Target>
      <LocURI>emailstore/1234</LocURI>
      <Filter>
        <Meta><Type>application/vnd.omads-email</Type></Meta>
        <Field>
          <Item>
            <Meta><Type>application/vnd.syncml-devinf+xml</Type></Meta>
            <Data>
```

```
              <Property>
                <PropName>emailitem</PropName>
                <PropParam>
                  <ParamName>texttype</ParamName>
                </PropParam>
                <PropParam>
                  <ParamName>attachtype</ParamName>
                  <ValEnum>transcode:image/png</ValEnum>
                </PropParam>
              </Property>
            </Data>
          </Item>
        </Field>
      </Filter>
    </Target>
  </Item>
</Get>
```

11.3 Getting a Specific Attached Object

Once a client has received a message with partial content, e.g. only the message headers and the attached objects' headers, the Get command must be used to retrieve the entire message record by sending the record LUID and Content-ID of the attached object within the Target LocURI (see [RFC2045] section 6.5.7).

Example 1: The following example illustrates how a client can retrieve a specific attached object of an email message that has already been synchronized; but only the email message headers and attached objects' headers have been synchronized. In this example the email message record LUID is 1234, and the Content-ID of the attached object is attach_id@mail.server.com.

```
<Get>
  <CmdId>5</CmdId>
  <Meta>
    <Type xmlns='syncml:metinf'>message/rfc2822</Type>
  </Meta>
  <Item>
    <Target>
      <LocURI>emailstore/1234/attach_id@mail.server.com</LocURI>
    </Target>
  </Item>
</Get>
```

Example 2: The following example, a variant of example 1, illustrates how a client can retrieve a specific attached object and request a transcoded version, e.g. image/png instead of the original image/jpeg.

```
<Get>
  <CmdId>5</CmdId>
  <Meta>
    <Type xmlns='syncml:metinf'>message/rfc2822</Type>
  </Meta>
  <Item>
    <Target>
```

```
      <LocURI>emailstore/1234/attach_id@mail.server.com</LocURI>
      <Filter>
        <Meta><Type>application/vnd.omads-email</Type></Meta>
        <Field>
          <Item>
            <Meta><Type>application/vnd.syncml-devinf+xml</Type></Meta>
            <Data>
              <Property>
                <PropName>emailitem</PropName>
                <PropParam>
                  <ParamName>attachtype</ParamName>
                  <ValEnum>transcode:image/png</ValEnum>
                </PropParam>
              </Property>
            </Data>
          </Item>
        </Field>
      </Filter>
    </Target>
  </Item>
</Get>
```

11.4 Smart Forward of an Email Message

The Smart Forward use case describes the case where the client is only requesting email message headers, including the attached objects, and then is forwarding the email message without having to download the full message content and/or all the attachments before. This is the responsibility of the server to re-attach the original entitities to the forward message when received from the client. The solution here is to use, within the [RFC2045] data, the Content-ID header field to reference specific or all entities of the original email message. The forwarded email message is a new message as-per [RFC2045], that is a new record sent to the server.

Example 1a: The following example illustrates how a client can forward a full email message that has already been synchronized; but only the email message headers and attached objects' headers have been synchronized. Only the email message content is shown here, i.e. placed within the Data element. The following forwarded email message contains its own message body and a reference to the original email message. Once the server receives this record, it can build the final email message by attaching the referenced entity.

```
Message-Id: <2007040900289@mail.server.com>
From: Jane Doe <jane.doe@server.com>
To: Jack Doe <jack.doe@server.com>
Date: Mon, 9 Apr 2007 20:06:07 -0800 (PST)
Subject: Fw: Example of email with attachment
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=some_boundary

--some_boundary
Content-Type: text/plain; charset=us-ascii
Content-ID: <forward_body_id@mail.server.com>

Hi Jack :

I am forwarding to you the email from John which contains the picture of
the OMA DS WG at a local restaurant.
```

```
--some_boundary
Content-Type: message/external-body;
              access-type=mail-server;
              server=mail.server.com

Content-Type: message/rfc2822
Content-ID: <2007040900069@mail.server.com>
```

Example 1b: The following example illustrates the resulting full email message reconstructed on the server, and how it could appear when displayed to the end user. Note that this includes the new message from Jane to Jack and the entire original email sent from John to Jane including the attachment.

```
From: Jane Doe <jane.doe@server.com>
To: Jack Doe <jack.doe@server.com>
Date: Mon, 9 Apr 2007 20:06:07 -0800 (PST)
Subject: Fw: Example of email with attachment

Hi Jack :

I am forwarding to you the email from John which contains the picture of
the OMA DS WG at a local restaurant.

-------- Original Message --------


From: John Doe <john.doe@server.com>
To: Jane Doe <jane.doe@server.com>
Date: Mon, 9 Apr 2007 08:15:39 -0800 (PST)
Subject: Example of email with attachment

Hi Jane :
Here is a picture of the OMA DS WG at a local restaurant.

OMADSWGimage.jpg
```

Example 2: The following example illustrates how a client can forward specific entities, like an attached object, of an email message that has already been synchronized. Shown here is the email message content, i.e. placed within the Data element. The following forwarded email message contains its own message body and the reference of one of the original attached objects. Once the server receives this, it can build the final email message by attaching the referenced entity.

```
Message-Id: <2007040900289@mail.server.com>
From: Jane Doe <jane.doe@server.com>
To: Jack Doe <jack.doe@server.com>
Date: Mon, 9 Apr 2007 20:06:07 -0800 (PST)
Subject: Fw: Example of email with attachment
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=some_boundary
```

```
--some_boundary
Content-Type: text/plain; charset=us-ascii
Content-ID: <forward_message_id@mail.server.com>


Hi Jack:
I am forwarding to you the picture of the OMA DS WG at a local restaurant
which John sent to me in an email.

--some_boundary
Content-Type: message/external-body;
              access-type=mail-server;
              server=mail.server.com;
              name="OMADSWGimage.jpg"

Content-Type: image/jpeg
Content-ID: <attach_id@mail.server.com>
```

Example 2b: The following example illustrates the resulting full email message that was reconstructed on the server, and how it could appear when displayed to the end user. Note that this includes the new message from Jane to Jack and ONLY the picture that was in the original email sent from John to Jane.

```
From: Jane Doe <jane.doe@server.com>
To: Jack Doe <jack.doe@server.com>
Date: Mon, 9 Apr 2007 20:06:07 -0800 (PST)
Subject: Fw: Example of email with attachment

Hi Jack :

I am forwarding to you the picture of the OMA DS WG at a local restaurant
which John sent to me in an email.


OMADSWGImage.jpg
```

11.5 Smart Reply of an Email Message

The Smart Reply use case describes the case where the client has synchronized part of a message, and then is replying to the received email message. By referencing the original message in the reply, the client is requesting the server to re-attach the original entities to the reply message. The solution here is to use, within the [RFC2045] data, the Content-ID header field to reference specific or all entities of the original email message. The reply email message is a new record sent to the server.

Example 1a: The following example illustrates how a client can reply to a message that has already been synchronized; but without transferring the entire message. Shown here is the email message content, i.e. placed within the Data element. The following reply email message contains its own message body and a reference to the original email message body content without any attachment. In this example, the To: field is left out under the assumption that the server will populate it based upon the referenced message. Once the server receives this, it can build the final email message by attaching the referenced entity and populating the To: header field.

```
Message-Id: <2007040900289@mail.server.com>
In-Reply-To: <2007040900069@mail.server.com>
References: <2007040900069@mail.server.com>
From: Jane Doe <jane.doe@server.com>
Date: Mon, 9 Apr 2007 20:06:07 -0800 (PST)
Subject: Re: Example of email with attachment
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=some_boundary

--some_boundary
Content-Type: text/plain; charset=us-ascii
Content-ID: <reply_message_id@mail.server.com>

Dear John:
Thanks for sending the picture of the OMA DS WG at dinner.  The next time
I am at an OMA meeting I will join you at the restaurant.

--some_boundary
Content-Type: message/external-body;
              access-type=mail-server;
              server=mail.server.com

Content-Type: text/plain
Content-ID: <body_id@mail.server.com>
```

Example 1b: The following example illustrates the resulting full email message that was reconstructed on the server, and how it could appear when displayed to the end user. The reconstruction done on the server depends on the Content-ID references to the building blocks of the message.

```
From: Jane Doe <jane.doe@server.com>
To: John Doe <john.doe@server.com>

Date: Mon, 9 Apr 2007 20:06:07 -0800 (PST)
Subject: Re: Example of email with attachment

Dear John:
Thanks for sending the picture of the OMA DS WG at dinner.  The next time
I am at an OMA meeting I will join you at the restaurant.

_____________________________________________________________________
From: John Doe <john.doe@server.com>
To: Jane Doe <jane.doe@server.com>
Date: Mon, 9 Apr 2007 08:15:39 -0800 (PST)
Subject: Example of email with attachment

Hi Jane :
Here is a picture of the OMA DS WG at a local restaurant.
```

Appendix A. Change History (Informative)

A.1 Approved Version History

Reference	Date	Description
OMA-TS-DS_DataObjEmail-V1_2-20060710-A	10 Jul 2006	Approved by TP ref#OMA-TP-2006-0239R03-INP_DS_V1_2_for_final_approval
OMA-TS-DS-DataObjEmail-V1_2_1-20070410-A	10 Apr 2007	Incorporated CRs: OMA-DS-DS_1_2-2006-0027 OMA-DS-DS_1_2-2007-0033R02 OMA-DS-DS_1_2-2007-0036
OMA-TS-DS-DataObjEmail-V1_2_1-20070810-A	10 Aug 2007	Prepared for TP notification TP ref # OMA-TP-2007-0326-INP_DS_V1_2_1_ERP_for_Notification