FUNAMBOL mobile open source

J2ME Mail API Design Document

October 2006

Changes History

Date	*Author*	*Changes*
09.2006	Giuseppe Monticelli	Document created
20.10.2006	Giuseppe Monticelli	Document reviewed
25.10.2006	Andrea Gazzaniga	Document reviewed
14.11.2006	Stefano Fornari	Minor changes due to document review
24.11.2006	Giuseppe Monticelli	More explications + changed UML class d.
12.09.2007	Marco Garatti	Description of changes applied to the Message implementation. (Mainly lazy/caching behavior)
28.11.2007	Ivano Brogonzoli	Description of usage for class Message, Folder and RMSStore updated
09.10.2008	Alessio Bianco	Added composedMessageLength field in the Message
21.05.2009	Carlo Codega	Review of the “Message storage and retrieval” section, to support the Multiple Email Account feature.

Table of Contents

1.Introduction....4
1.1. Overview....4
1.2. The Funambol Mail API....4
1.3. Mail API Architecture....4
1.3.1. Part....5
1.3.1.1. Examples....7
1.3.2. Message....10
1.3.3. BodyPart....11
1.3.4. Multipart....12
1.4. Message Storage and Retrieval....12
1.4.1. Folder....13
1.4.2. AccountFolder....13
1.4.3. The Interface 'Store' and its implementation 'RMSStore'....13
1.4.4. The RmsRecordItem Interface....14
1.5. Message Parsing and Formatting....14
1.5.1. MIMEParser and MIMEFormatter....14
2.Appendices....16
2.1. Appendix A – Open Issues....16
2.2. Appendix B – References....16

1. Introduction

1.1. Overview

In order to allow easy email clients development in Java for J2ME compliant micro devices, a new Java Mail API aimed at describing email objects as per RFC 2822 and the MIME specifications family (RFC 2045, 2046, 2047, 2048, 2049) is introduced. This API offers the possibility to save and retrieve email messages into and from the device's persistent store, disposing them in any folder hierarchy.

1.2. The Funambol Mail API

The approach is similar to (but not the same of) the design of the existing Sun JavaMail API [JAVAMAIL], in which a layered architecture allows clients to use the same Java API to send, receive and store email messages using different data-types from different message stores and using different message transport protocols.

1.3. Mail API Architecture

The class diagram of Figure 1.1 illustrates the architecture of the Funambol Mail API, where all classes belong to the com.funambol.mail package. The notation is UML 1.4 compliant (see [UML 1.4]; for a quick reference of the used UML symbols see [UML 2.0], in particular the chapter 7, *Classes*, and the related tables 7.2 and 7.3). To modify the picture, edit the *.zargo* file distributed with this document (using the free tool [ArgoUML]) and export the modified diagram as a graphical object:

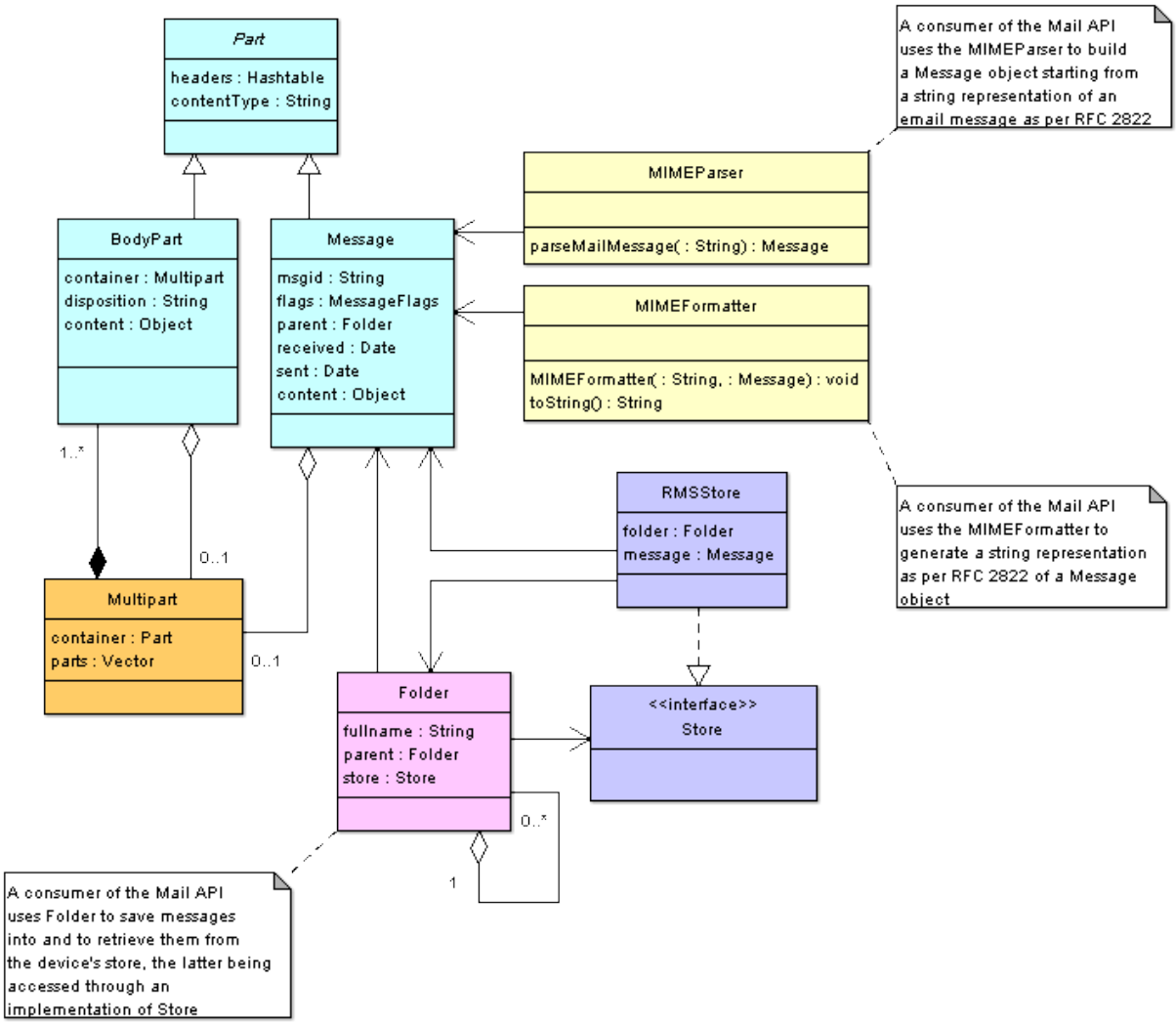

Figure 1.1: Funambol Mail API Class Diagram

The core of the architecture is the Part abstract class, which models what in the specifications ([RFC 2045], § 2.4) is defined as an *entity*, i.e. the combination of headers and a content body. A Part can be an email Message or, in case of multi-part email, a BodyPart. A Message containing at least a BodyPart needs a container to the all parts. This container is a Multipart. Note that a Multipart is not an entity. Every BodyPart can contain recursively a Multipart that can contain in turn other BodyParts (for an example of this case, see 1.3.1.1).

1.3.1. Part

This is the common base class for a single-part message and for each body part within a multi-part message (grouped together with other body parts in a Multipart container). A Part is built of a set of attributes (also called 'headers') followed by a content (also called 'body'). Note that a 'single-part' message (e.g. a message without attachments) does not contain a body part (a BodyPart object in the Java implementation), therefore retrieving the content of such a 'single-part' message is equivalent to retrieving the body as pure text. Examples of common header attributes, as defined in the [RFC 2822] specification (§ 2.2), are "To:", "From:", "Date:" etc. The header field "Content-Type" is defined among others attributes dealing with encoding and content transfer issues in the first MIME ("Multipurpose Internet Mail Extension") specification (see [RFC 2045], § 5).

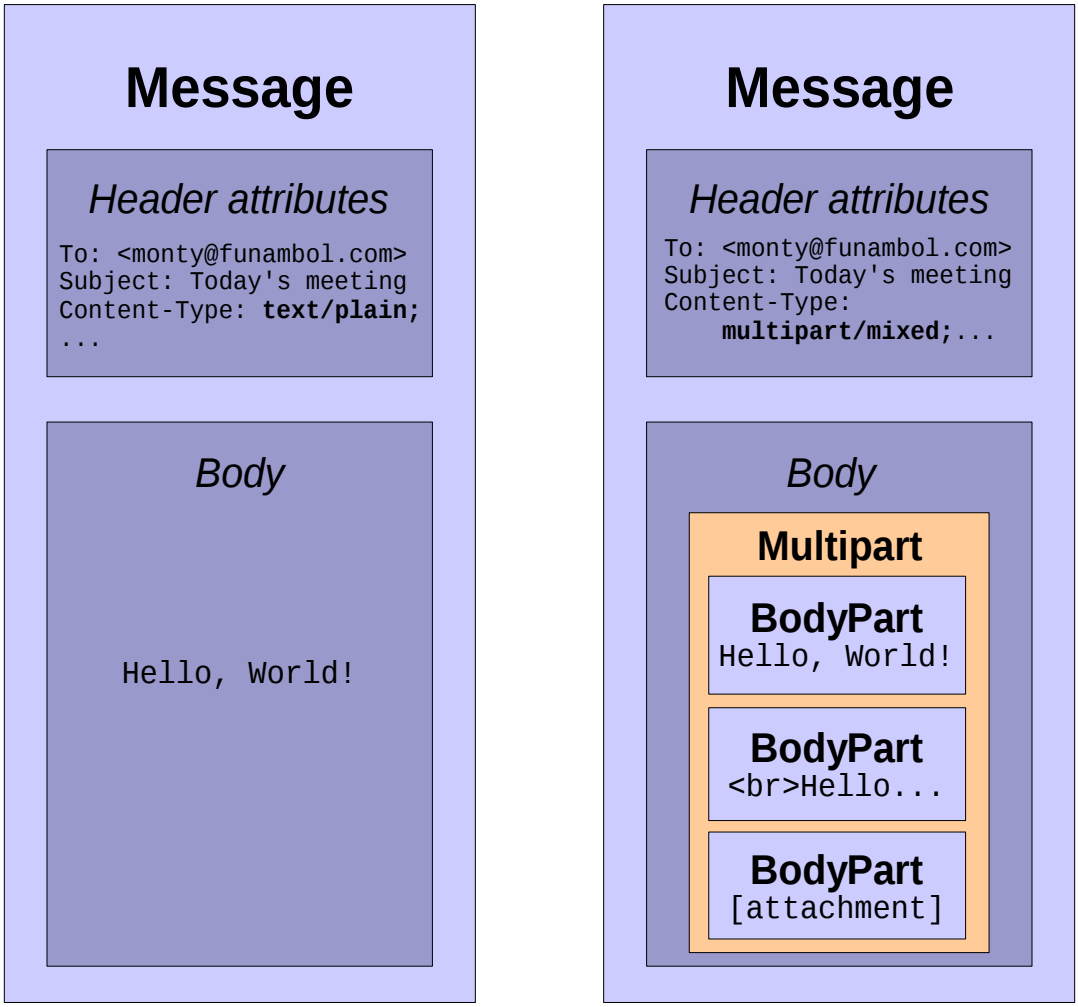

Figure 1.2: Differences between a 'single-part' and a multi-part Message

The abstract class Part provides access methods to the attributes that are required in order to define and format the data content carried by an email message (as per [RFC 2822]). Note that only the class representing an email message (Message) provides fields corresponding to the usual email headers such as "From:", "To:", "Subject:", "CC:" etc. Multipart instead, is not an extension of Part and doesn't directly contain headers; this is because a Multipart object is always treated as the content of the body of a Message or BodyPart object. Headers are represented as name-value pairs, where both the name and value are Strings. A BodyPart can contain recursively other body parts included in a nested Multipart object.

While dealing with the Content-Type header attribute, the MIME typing system is used (see [RFC 2046], in particular section 3) that defines seven standard media types for describing complex email messages. Five types are discrete (text, image, audio, video, application), two are composite (multipart, message). To these top-level media types a series of subtypes are to be added, indicated in the header attributes as values bounded to the top-level types through a slash ([top-level type]/[subtype]. So, for example, very usual combinations of top-level media types and subtypes are text/plain, text/html, image/jpeg for the discrete types, and multipart/alternative, multipart/mixed, message/rfc822 for the composite types.

The content is returned only as an Object through the method getContent(). This means that the returned object is of course dependent on the content itself. In particular, a multi-part Part's content (i.e. the content of a multi-part Message) is always a Multipart object, whereas the content of a 'single-part' Message with content type equal to "text/plain" is always a text String.

1.3.1.1. Examples

Here is an example of a forwarded email message: the original was a multipart/mixed message (because it was constituted by a text in the plain version and in the HTML version plus a binary attachment) and had the following structure:

```
Message-ID: <c96b5e8f0611240541o3380cbb4m457eed3918e88d02@mail.gmail.com>
Date: Fri, 24 Nov 2006 14:41:34 +0100
From: "Michael Schumacher" <micha.schumi@googlemail.com>
To: "Roberto Baggio" <baggio@funambol.com>
Subject: Rich formatting with binary attachment
MIME-Version: 1.0
Content-Type: multipart/mixed;
       boundary="----=_Part_95258_25174283.1164375694237"

------=_Part_95258_25174283.1164375694237
Content-Type: multipart/alternative;
       boundary="----=_Part_95259_16350681.1164375694237"

------=_Part_95259_16350681.1164375694237
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: 7bit
Content-Disposition: inline

Rich formatted text! :-)

--
Michael Schumacher
Trient-Allee 26
A - 26845 Quittenburg

------=_Part_95259_16350681.1164375694237
Content-Type: text/html; charset=ISO-8859-1
Content-Transfer-Encoding: 7bit
Content-Disposition: inline

<span style="color: rgb(255, 0, 0); font-weight: bold;">Rich formatted text!
</span>:-)<br><br>-- <br>Michael Schumacher<br>Trient-Allee 26<br>A - 26845 Quittenburg<br>

------=_Part_95259_16350681.1164375694237--

------=_Part_95258_25174283.1164375694237
Content-Type: application/octet-stream; name="demo.license"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="demo.license"
X-Attachment-Id: f_euwn7o4e

TmFtZT1HaXVzZXBwZSBNb250aWNlbGxpCkVtYWlsPWdpdXNlcHBlLm1vbnRpY2VsbGlAZ29vZ2xl
bWFpbC5jb20KRGVtb1VudGlsPTIwMDYtMTItMjMKS2V5PTJnUC08OWV2Ri1mekRaTC1uMmFNMS1t
M1J6UC15bUkzPi00aTw3dS1HOFhTdC1PVnp5RAoK
```

```
------=_Part_95258_25174283.1164375694237--
```

In the following the names of the Java classes are used to describe the corresponding parts of the email message. In the header attribute 'Content-Type' of the Message is indicated the MIME type of its content: multipart/mixed. Therefore the content of the Message is a Multipart consisting of two BodyParts (the parts between the blue boundaries "------=_Part_**95258**_25174283.1164375694237" and "------=_Part_**95258**_25174283.1164375694237**--**" (notice the two dashes indicating that that boundary is the last of the Multipart object). The first BodyPart contains in turn, as stated in its header attribute 'Content-Type' (multipart/alternative), a Multipart. This Multipart consists of two BodyParts: the first of MIME type text/plain, the second of type text/html. They are contained between the green boundaries "------=_Part_**95259**_16350681.1164375694237" and "------=_Part_**95259**_16350681.1164375694237**--**" (again, notice the two dashes indicating the last boundary). Returning to the first two BodyParts (the blue ones), the second is the attachment, in particular of MIME type application/octet-stream (it is a binary file, hence it was encoded with the Base64 algorithm as indicated in the corresponding header attribute of this BodyPart).

Once forwarded (with a different mail client and together with the original attachment), the original message above assumes the following form:

```
Message-ID: <45671B1F.8040705@funambol.com>
Date: Fri, 24 Nov 2006 17:17:35 +0100
From: Roberto Baggio <baggio@funambol.com>
MIME-Version: 1.0
To: Michael Schumacher <micha.schumi@googlemail.com>
Subject: [Fwd: Rich formatting with binary attachment]
Content-Type: multipart/mixed;
 boundary="------------020302060009070003040203"

This is a multi-part message in MIME format.
--------------020302060009070003040203
Content-Type: multipart/alternative;
 boundary="------------000604000005090504060808"


--------------000604000005090504060808
Content-Type: text/plain; charset=ISO-8859-1
Content-Transfer-Encoding: 7bit



-------- Original Message --------
Subject:       Rich formatting with binary attachment
Date:   Fri, 24 Nov 2006 14:41:34 +0100
From:   Michael Schumacher <micha.schumi@googlemail.com>
To:     Roberto Baggio <baggio@funambol.com>



Rich formatted text! :-)

--
Michael Schumacher
Trient-Allee 26
```

```
A - 26845 Quittenburg



--
Roberto Baggio

 funambol :: mobile open source :: http://www.funambol.com


--------------000604000005090504060808
Content-Type: text/html; charset=ISO-8859-1
Content-Transfer-Encoding: 7bit

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
</head>
<body bgcolor="#ffffff" text="#000000">
<br>
<br>
-------- Original Message --------
<table class="moz-email-headers-table" border="0" cellpadding="0"
 cellspacing="0">
  <tbody>
    <tr>
      <th align="right" nowrap="nowrap" valign="baseline">Subject: </th>
      <td>Rich formatting with binary attachment</td>
    </tr>
    <tr>
      <th align="right" nowrap="nowrap" valign="baseline">Date: </th>
      <td>Fri, 24 Nov 2006 14:41:34 +0100</td>
    </tr>
    <tr>
      <th align="right" nowrap="nowrap" valign="baseline">From: </th>
      <td>Michael Schumacher <a class="moz-txt-link-rfc2396E"
href="mailto:micha.schumi@googlemail.com">&lt;micha.schumi@googlemail.com&gt;</a></td>
    </tr>
    <tr>
      <th align="right" nowrap="nowrap" valign="baseline">To: </th>
      <td>Roberto Baggio <a class="moz-txt-link-rfc2396E"
href="mailto:baggio@funambol.com">&lt;baggio@funambol.com&gt;</a></td>
    </tr>
  </tbody>
</table>
<br>
<br>
<pre>Rich formatted text! :-)
```

```
--
Michael Schumacher
Trient-Allee 26
A - 26845 Quittenburg


</pre>
<br>
<pre class="moz-signature" cols="72">--
Roberto Baggio


 funambol :: mobile open source :: <a class="moz-txt-link-freetext"
href="http://www.funambol.com">http://www.funambol.com</a></pre>
</body>
</html>


--------------000604000005090504060808--


--------------020302060009070003040203
Content-Type: application/octet-stream;
 name="demo.license"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
 filename="demo.license"

TmFtZT1HaXVzZXBwZSBNb250aWNlbGxpCkVtYWlsPWdpdXNlcHBlLm1vbnRpY2VsbGlAZ29v
Z2xlbWFpbC5jb20KRGVtb1VudGlsPTIwMDYtMTItMjMKS2V5PTJnUC08OWV2Ri1mekRaTC1u
MmFNMS1tM1J6UC15bUkzPi00aTw3dS1HOFhTdC1PVnp5RAoK
--------------020302060009070003040203--
```

Notice that the structure of the message hasn't changed, and that the embedded, forwarded message was sent again as a "multipart" and not as a "message". Normally, the use of the MIME type message/rfc822 is very uncommon.

1.3.2. Message

This is a concrete class modeling a MIME message. It extends the Part abstract class.

In addition to what provided to Part, a Message has the properties listed in the table below.

Property	***Description***
flags	A combination of status flags associated with this Messager, e.g. 'read', 'forwarded', 'draft' etc. This is an object of the class MessageFlags
parent	A Folder can contain many Messages, and each Message has a reference to its parent Folder
sent	The 'sent' date of this Message (a Date object)
received	The 'received' date of this Message (a Date object)
content	The content of this Message as an Object: it can be a Multipart or a String. When it is a String, it is stored in the native encoding of the application, regardless of how it has been transferred. During parsing operation then the String is decoded according to the 'Content-Transfer-Encoding' field
laziness	A bitmask specifying the message behavior for lazy loading

Property	*Description*
	(see below for more details)
msgid	The unique ID of this message (got from email headers)
key	The client unique ID for this message. The client is responsible for setting this information. The information is not used in the API, therefore the client is free to define it to any value or even leave it undefined.
recordId	This information may be used to speed up the store/retrieve of the message in a Store that is positional. For example in the RMSStore this is set to the RMS record id. It is up to the **unique** store that holds the message to set this information if it makes sense.
composedMessageLength	The length of the composed message. Is set to distinguish the replied/forwarded message from the new message.

In order to let Message objects to be serialized and stored in the device's persistent store, this class implements the Serializable interface from the package com.funambol.storage (see [FUNAMBOLCOMMON]).

Message can be configured to be more or less memory hungry. This is done by setting the laziness levels. Each message can handle its content and headers in a lazy way. This means that this information is loaded from the store containing the message (if any) whenever needed and flushed immediately afterward. A message with lazy content and headers has a small memory footprint but tends to be slow, especially if accesses to the store holding the message are slow. To mitigate this problem it is possible to configure messages to cache some individual values that are in the headers. For example it is possible to cache the subject, the list of “to” recipients, the “from” and so on. Memory consumption worsen when caching is enabled, but caching only some header items is still significantly better than keeping all the headers in memory.

Message has a global default behavior about what information is kept lazy and what is cached. This global behavior is applied to all the messages, but it is possible to configure individual message to force a different behavior.

The valid values for laziness are: NO_LAZY, LAZY_CONTENT and LAZY_HEADERS (they can be combined).

The valid values for caching are: CACHE_SUBJECT, CACHE_FROM, CACHE_TO, CACHE_CC, CACHE_BCC, CACHE_REPLYTO. These values can be combined together as well.

1.3.3. BodyPart

A BodyPart object is designed to be inserted into a Multipart container which, in turns, is contained in a multi-part Message having MIME media type equal to "multipart". The class BodyPart, like Message, extends the abstract class Part.

As illustrated in Figure 1.3, a BodyPart of type “text/plain” contains a headers section and a text body; a BodyPart of type “multipart” contains a header section and another Multipart.

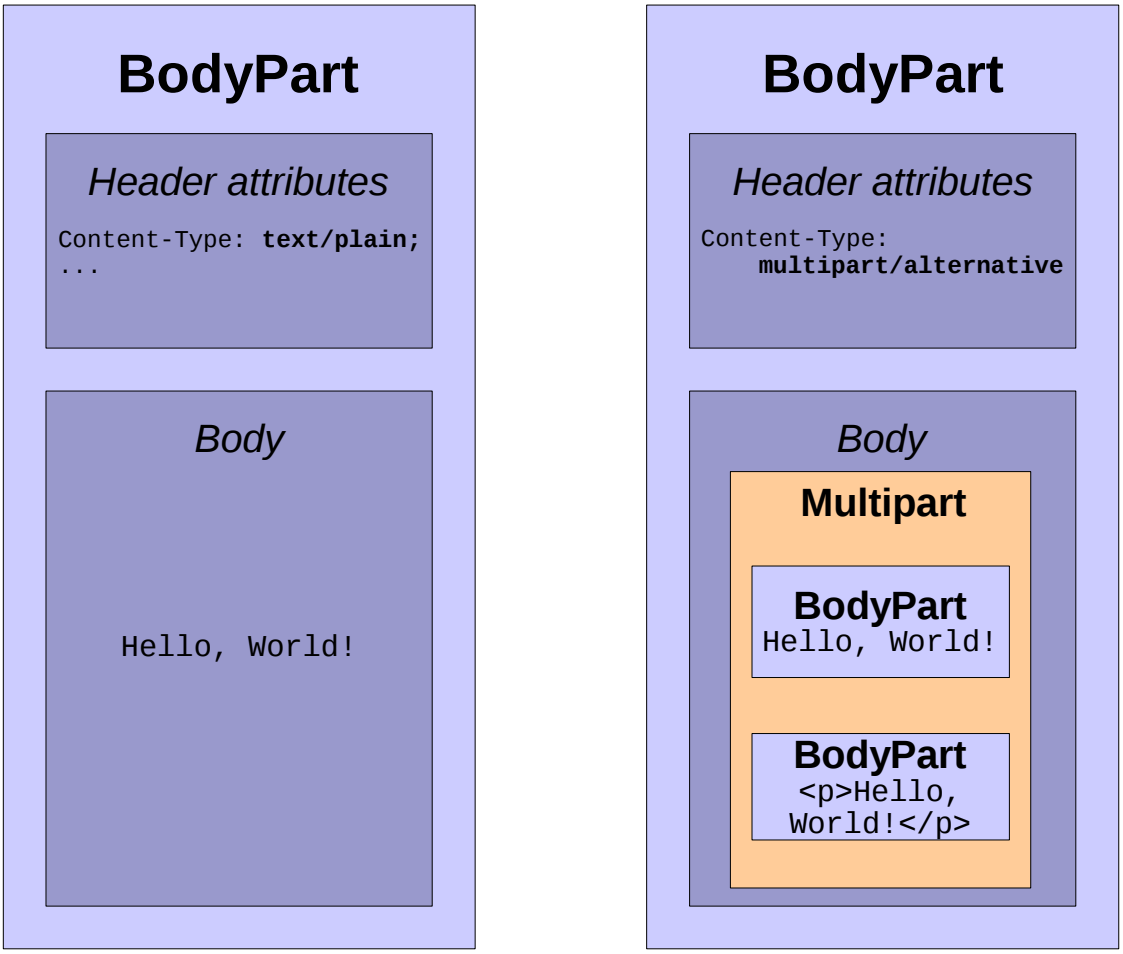

Figure 1.3: A BodyPart with attributes and its content body when content type is text or multipart

1.3.4. Multipart

This class is a container that contains objects of type BodyPart. Note that this is not an extension of the abstract class Part, since it is not a part in itself.

A Multipart has the properties defined in the table below.

Property	***Description***
container	The Message or BodyPart eventually containing this Multipart
parts	A Vector representing the list of the BodyParts contained in this Multipart

1.4. Message Storage and Retrieval

A Message isn't just an entity being carried as a stream of bytes over a connection. Furthermore, a Message is something that once received by a user agent has to be physically stored into a proper mail folder belonging to the related Mail Account, in order to grant to the user at any time within a session the possibility to read and forward it, or to respond to it.

The Funambol Mail API provides a hierarchical system that can be used to store and retrieve Message items. Messages can be organized in folders as you can do with a typical file system, through the Store, Folder and AccountFolder objects:

•Store provides the interface to directly handling from the hierarchical store (e.g. get the list of folders, add/remove folders, get/add/remove messages, etc.). (See 1.4.3)

•Folder is a collection of items like Messages, Folders or both. (See 1.4.1)

•AccountFolder is a particular Folder which represents a Mail Account. (See 1.4.2)

1.4.1. Folder

This class represents a mailbox folder in the device storage and models a node in the message storage hierarchy used to organize messages. Folders can contain Message objects, other Folder objects, or both.

Folder is the object used by other consumer-objects to access Messages stored in the device's permanent store or to actively save these into this store, and to handle a folder item itself. If you want to create or delete other Folders you should use the Store object directly.

A Folder is characterized by the following attributes:

•fullname: the complete path to the folder (e.g. /ParentFolder/Subfolder);

•role: the role of the Folder (e.g. 'Inbox', 'Outbox', etc.);

•created: the creation date of the Folder;

•store: the reference to the underlying RMS store.

Further attributes are: the reference to the parent Folder and one to the children. They can be set dynamically.

The opportunity to define listeners to the folders is under discussion.

1.4.2. AccountFolder

This class represents a Mail Account, which is particular Folder characterized by two additional attributes:

•visibleName: the visible name of the account;

•emailAddress: the email address of the account.

The role of this particular folder is fixed to “account”, in order to differentiate it from normal folders.

Usually this kind of folder should contain only folders representing the Mail Account folders, like Inbox and Outbox. By the way it's prepared to contain Message items also.

1.4.3. The Interface 'Store' and its implementation 'RMSStore'

The interface Store models the messages database and the access protocol used to manage the message store. Using the Store object you can add, update and remove Message and Folder items (including AccountFolder).

Two implementations are possible within the Funambol Mail API, each to address a different access to the persistent memory of a J2ME compliant device. A JSR-75 compliant implementation (writing in the file system hierarchy of the device: see [CONNFRAMEWORK]) wouldn't be currently possible on every device. The class RMSStore addresses the standard J2ME management of the device's store provided by the J2ME Record Management System (RMS).

The RMSStore uses an ObjectStore instance to interact directly with the underlying data store (see Funambol Common API module). Each Folder refers to a specific RecordStore, which can contain different record (called child in the RMSStore implementation) types, differentiated by a prefix character (see 1.4.4): Messages and/or other Folders. A Folder record (contained in a Folder RecordStore) includes the reference to the related RecordStore, that is the path of the subfolder. It means that all the Folders contained in the device's store has a RecordStore associated with, and its parent Folder contains the reference to it. The only Folder which doesn't belong to any other, is the root Folder (“/”).

The following picture aims to describe better the RMSStore system:

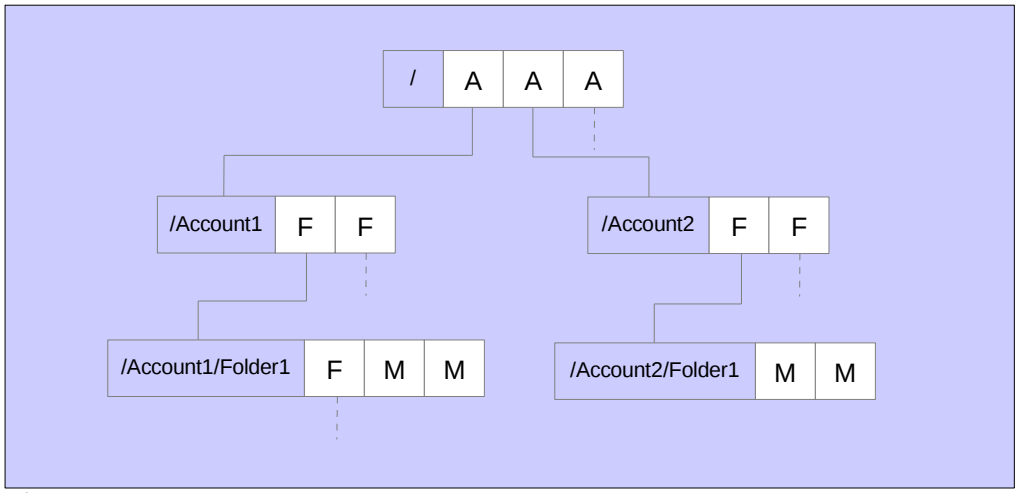

Figure 1.4: RMSStore

Each box represents a RecordStore/Folder, identified by the path. The records (the white boxes) are identified by the prefix character.

1.4.4. The RmsRecordItem Interface

Objects that need to be stored need to be serialized in order to be added as bytes arrays to the device's RMS (implemented by the class javax.microedition.rms.RecordStore of the J2ME's MIDP profile). Such objects (Folder, AccountFolder, Message) implement the RmsRecordItem interface of the Funambol Common API.

A RmsRecordItem extends the Serializable interface and must also be able to handle the record id.

In order to differentiate the content type of the RMS records, the Serializable objects must append an identifier prefix at the beginning of the record byte stream (e.g. through the serialize() method). Such prefixes are defined as a 2 bytes char, for different objects used in the Funambol Mail API:

•'M' for Message items;

•'F' for Folder items;

•'A' for AccountFolder items.

This prefix is used by the Store implementation to understand the content type, and should not be read by the RmsRecordItem deserialize() method.

Other than in the Java 2 Standard Edition (J2SE) the serialization of objects isn't supported by J2ME. This functionality is also provided by the Funambol API mostly with the class com.funambol.storage.ComplexSerializer belonging to the same package as Serializable. The description of this package is in [FUNAMBOLCOMMON].

Due to the reduced storage capabilities of J2ME compliant devices, an intelligent rotation policy of stored messages has to be implemented, in order to permit to new incoming messages to be stored in place of older messages or, by converse, to prevent older messages to be overwritten by new incoming messages. How to configure this policy has to be demanded to the user.

1.5. Message Parsing and Formatting

1.5.1. MIMEParser and MIMEFormatter

The format of an email message is defined by a group of RFCs: [RFC 2822], [RFC 2045], [RFC 2046]).

MIMEParser is responsible for parsing a string containing a message in the format specified by RFC 2822 with MIME extensions into a Message object. The parser supports the following algorithms for decoding encoded contents:

- Quoted printable
- Base64

The headers handled by the current implementation of MIMEParser are described in the table below:

Header	***Description***
Subject:	The subject of the email message
From:	The sender of this email message
To:	The addressee of this email message
Date:	The sent date of the message
Message-ID:	The unique ID of the message
User-Agent:	The mail client used to send the message
MIME-Version	The version of the MIME specification (currently "1.0")
Content-Type:	The MIME media type/subtype of the content + original character set
Content-Transfer-Encoding:	The algorithm used to encode the original character set

MIMEFormatter, instead, converts a Message object into a text MIME based message (again, this is a string containing the message as a text, ready to be passed to the application's transmission layer). Because of the low resources available on mobile devices, not all the headers contained in the incoming message are stored, so the process to parse and then to format again a message is not necessary symmetric. The header attributes supported at present by the MIMEFormatter are resumed in the following table:

Header	***Description***
Subject:	The subject of the email message
From:	The sender of this email message
To:	The addressee of this email message
Date:	The sent date of the message
Message-ID:	The unique ID of the message
User-Agent:	The mail client used to send the message
MIME-Version	The version of the MIME specification (currently "1.0")
Content-Type:	The MIME media type/subtype of the content + original character set
Content-Transfer-Encoding:	The algorithm used to encode the original character set

2. Appendices

2.1. Appendix A – Open Issues

The following table contains questions and answers on still open issues.

Q		
A		

2.2. Appendix B – References

[JAVAMAIL] Sun Microsystems, JavaMail API Design Specification v. 1.4, December 2005

[RFC 2045] Network Working Group, Multipurpose Internet Mail Extensions (MIME). Part One: Format of Internet Message Bodies (http://www.ietf.org/rfc/rfc2045.txt)

[RFC 2046] Network Working Group, Multipurpose Internet Mail Extensions (MIME). Part Two: Media Types (http://www.ietf.org/rfc/rfc2046.txt)

[RFC 2822] Network Working Group, Internet Message Format (http://www.ietf.org/rfc/rfc2822.txt)

[FUNAMBOLCOMMON] Funambol, Funambol J2ME Common API Design Document, October 2006

[UML 2.0] Object Management Group, Unified Modeling Language: Superstructure. Version 2.0, August 2005 (http://www.omg.org/cgi-bin/doc?formal/05-07-04)

[ArgoUML] ArgoUML. Download and documentation under http://argouml.tigris.org/

[UML 1.4] OMG Unified Modeling Language Specification. Version 1.4, September 2001 (http://www.omg.org/cgi-bin/doc?formal/01-09-67)

[CONNFRAMEWORK] Sun J2ME Generic Connection Framework (JSR75 - File Connection API): http://www.j2medev.com/api/fileconnection/javax/microedition/io/file/package-summary.html