JSR 205 WMA 2.0

Version 0.9, Draft

API GUIDE

COPYRIGHT

Trademarks and Service Marks

About This Document

This document covers in brief about Wireless Messaging API 2.0, it includes overview of WMA 2.0 Architecture and APIs, followed by a sample code snippet on send and receive MMS.

Scope

This document is intended for Java ME developers who wish to develop Java ME applications. It assumes good knowledge of java programming language.

Document History

Date	Version	Comment
10/11/09	0.9	Draft

References

- WMA 2.0 Specification

  http://jcp.org/en/jsr/detail?id=205

- WMA 2.0 API:

  http://developers.sun.com/mobility/midp/articles/wma2

Abbreviations

Java ME	Java Platform Micro Edition
CLDC	Connected Limited Device Configuration
WMA	Wireless Messaging API
CBS	Cell Broadcast Service
SMS	Short Messaging Service
CGF	Generic Connection Framework
URL	Uniform Resource Locator

Table of Contents

Table of Figures

Introduction

The Wireless Messaging API 2.0 (WMA 2.0) is an optional package for the Java Platform Micro Edition (Java ME). It is used for platform independent access to wireless communication resources like Short Message Service (SMS), Cell Broadcast Service (CBS) and the Multimedia Messaging Service (MMS). It is mainly used for sending MMS which includes audio, text, image and video. The messaging API is based on the Generic Connection Framework (GCF), which is defined in the Connected Limited Device Configuration (CLDC) specification.

Initially WMA is introduced in the Java ME as (WMA 1.1) JSR 120; WMA 1.1 has been enhanced and released as WMA 2.0 (JSR 205).

The differences between WMA versions 1.1 and 2.0 are related primarily to the support of multi-part messages used for MMS messaging.

Packages

WMA 2.0 has the following packages:

javax.microedition.io : This package includes the platform networking interfaces which have been modified for use on platforms that support message connections.

javax.wireless.messaging: This package defines an API which allows applications to send and receive wireless messages.

javax.microedition.io

This package includes the platform networking interfaces which have been modified for use on platforms that support message connections. It contains the Connector class and contains SecurityException as an exception return from calls to open().

javax.wireless.messaging

This package defines an API which allows applications to send and receive wireless messages. The API is generic and independent of the underlying messaging protocol.

This package includes the below mentioned interfaces and classes.

BinaryMessage: interface, which is useful for Binary messages.

Message: This interface is the base interface for derived interfaces that represent various types of messages.

MessageConnection: interface, which is useful for sending and receiving messages.

MessageListener: interface, useful to notify the incoming messages.

MultipartMessage: interface, useful for multipart messages.

TextMessage: interface, useful for the text message

MessagePart: this class is useful for adding the instance of the messagePart to MultipartMessage.

WMA 2.0 Message Types

WMA 2.0 defines four types of message representations. Additionally, it defines a *message part* in support of multipart messages, used for carrying multimedia messages.

The Message Interface

The interface javax.wireless.messaging.Message is the base type for all messages communicated using WMA 2.0 - A Message contains source address, destination addresses, and a payload.

The below mentioned methods are used to get and set the message's source and destination addresses, and to get its timestamp:

- String getAddress();
- void setAddress(String address);
- Date getTimestamp();

WMA 2.0 defines three sub interfaces of Message, as shown in Figure 1:

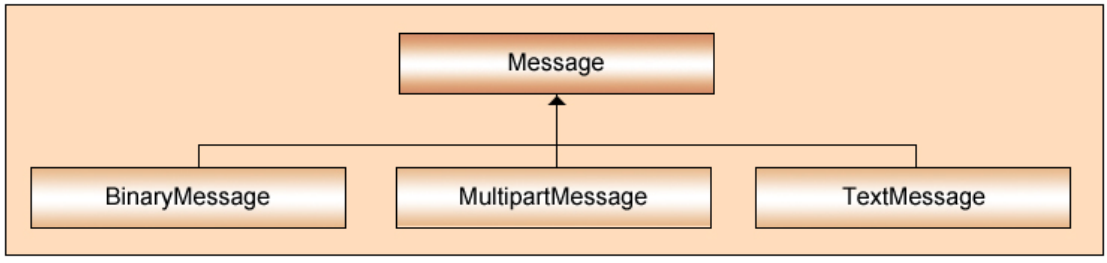

Figure 1: The Message Interface and its Subinterfaces

BinaryMessage

BinaryMessage subinterface represents a message with a binary payload, which can be sent as an SMS-based short binary message. This interface declares methods to set and get the binary payload as an array of bytes:

- byte[] getPayloadData();
- void setPayloadData(byte[] bytes);

Methods to set and get the address of the message, and to get its timestamp, are all inherited from Message.

TextMessage

TextMessage subinterface represents a message with a text payload, which can be sent as an SMS-based short text message. This interface provides methods to set and get the text as an instance of String:

- String getPayloadText();
- void setPayloadText(String data);

The methods to set and get the address of the message and get its timestamp are inherited from Message.

MultipartMessage

MultipartMessage subinterface as the name suggests represents a message that consists of multiple parts, mainly an MMS-based multimedia message. It defines a combination of one or more MessageParts, and provides methods to manage the sender and recipient addresses, message's headers, "start message" content ID, and the message's parts: The methods available are:

- boolean addAddress(String type, String address);
- void addMessagePart(MessagePart messagePart) throws sizeExceededException;
- String getAddress();
- String[] getAddresses(String type);
- String getHeader(String headerField);
- MessagePart getMessagePart(String contentID);
- MessagePart[] getMessageParts();
- String getStartContentId();
- String getSubject();
- boolean removeAddress(String type, String address);
- void removeAddresses();
- void removeAddresses(String type);
- boolean removeMessagePart(MessagePart messagePart);
- boolean removeMessagePartId(String contentID);
- boolean removeMessagePartLocation(String contentLocation);
- void setAddress(String address);
- void setHeader(String headerField, String headerValue);
- void setStartContentId(String contentID);

- void setSubject(String subject);

MultipartMessage overrides the methods to set and get the message's address it inherits from Message.

Multipart messages follow the format of standard emails, which consist of RFC822-based headers and multiple parts based on the Multipurpose Internet Mail Extensions (MIME) standard defined by the World Wide Web Consortium (W3C) in RFC2045 and RFC2046, as shown in Figure 2.

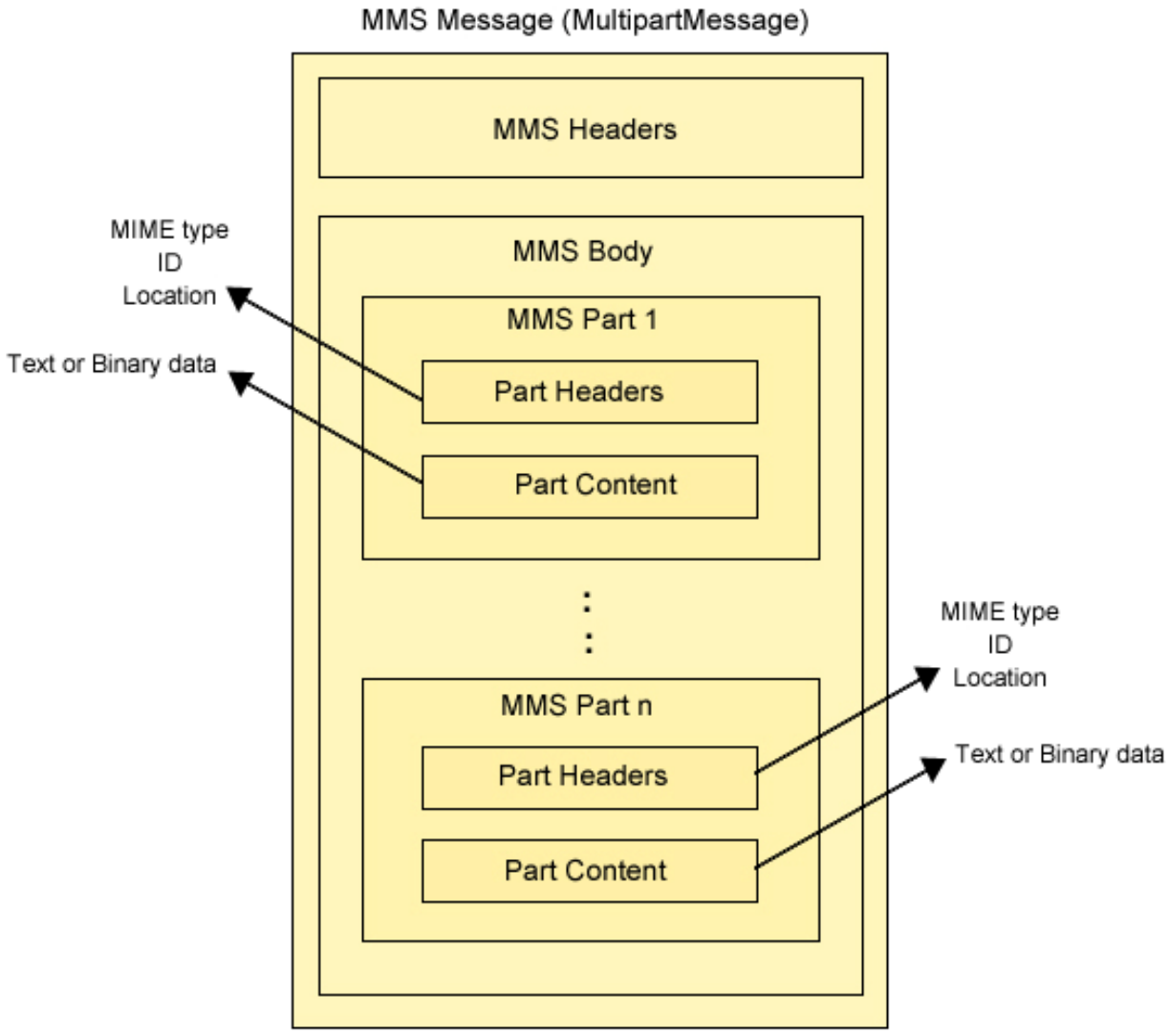

Figure 2: Structure of an MMS Multipart Message

The MultipartMessage interface represents the multimedia message body and its headers, while a MessagePart class instance represents each individual MIME part.

MessagePart

MessagePart represents one part of a message. Apart from having various constructors, this class provides methods to retrieve the content, and the related information about the content. The methods available are:

- MessagePart(byte[] contents, int offset, int length, String mimeType, String contentId, String contentLocation, String encoding) throws SizeExceededException;
- MessagePart(byte[] contents, String mimeType, String contentId, String contentLocation, String encoding) throws SizeExceededException;
- MessagePart(java.io.InputStream is, String mimeType, String contentId, String contentLocation, String encoding) throws IOException, SizeExceededException;
- public byte[] getContent();
- public InputStream getContentAsStream();
- public String getContentID();
- public String getContentLocation();
- public String getEncoding();
- public int getLength();
- public String getMIMEType();

A message part consists of a MIME type as defined in RFC2046, a content ID, a content location, and the content itself.

MessageConnection

MessageConnection defines methods for creating TextMessages, BinaryMessages, and MultipartMessages, method to calculate the number of protocol segments needed for sending the message, methods to receive and send messages, and a method to set the message listener for this connection. The application opens a MessageConnection with the Generic Connection Framework by providing a URL connection string. The methods available are:

- Message newMessage(String type, String address);
- int numberOfSegments(Message message);
- Message receive() throws IOException, InterruptedIOException;
- void send(Message message) throws IOException, InterruptedIOException;
- void setMessageListener(MessageListener messageListener) throws IOException;

Also, the interface defines String constants, one of which used to identify the type of the message which needs to be created by using factory method newMessage (…).

- String TEXT_MESSAGE = "text_msg";
- String BINARY_MESSAGE = "binary_msg";
- String MULTIPART_MESSAGE = "multipart_msg";

When creating a message connection, always one of the above mentioned constants can be used.

```
…
mmsconnection = (MessageConnection)Connector.open(address);
MultipartMessage mmmessage =
MultipartMessage)mmsconnection.newMessage(MessageConnection.MULTIPART_ME
SSAGE);
mmmessage.setAddress(address);
mmsconnection.send(mmmessage);
…
```

MessageConnection can be opened and closed by the connection factory methods javax.microedition.io.Connector.open() and javax.microedition.io.Connection.close().

Push Registry

Sometimes there is a requirement that MIDlet should auto start whenever a new message arrives. For this PushRegistry is useful. The purpose of PushRegistry is to start the MIDlet automatically whenever the message is received.

Permissions

WMA uses the underlying platform's security framework like to open a connection or to send and receive messages; permissions must be requested by the application and granted by the platform. These are implemented depending upon the requirement of specification. In MIDP 2.0, permissions are requested by way of the JAD or the manifest, and granted by the user when the operations invoked. For signed MIDlets, permissions must be defined in the manifest.

Below are some of the class and methods where SecurityException is thrown.

- javax.microedition.io.Connector: if the application is not granted permission to create a connection for a given messaging protocol, as defined by the platform security services
- MessageConnection.send(): if the application has no permission to send messages on the specified port
- MessageConnection.receive(): if the application has no permission to receive messages on the specified port

Below example shows WMA permissions requests via the JAD or manifest:

```
...
MIDlet-Permissions: javax.microedition.io.Connector.sms,
javax.wireless.messaging.sms.receive,
javax.wireless.messaging.sms.send,
javax.microedition.io.Connector.cbs,
javax.wireless.messaging.cbs.receive,
javax.microedition.io.Connector.mms,
javax.wireless.messaging.mms.receive,
javax.wireless.messaging.mms.send,
javax.microedition.io.PushRegistry
…
```

binary message to and from mobile phone originally defined as part of the GSM series of standards.

Maximum length of an each message is 160 characters.

The base interface that is implemented by all messages is named as javax.wireless.messaging.Message. It provides methods for addresses and timestamps.

Message interface provides methods that are common for all messages.

The data part of the message consists of both text message and binary message, which are represented by TextMessage and BinaryMessage interfaces which are derived from Message. The message sending and receiving functionality is implemented by a MessageConnection (derived from Connection interface of Generic Connection Framework (GCF)). The methods for sending and receiving messages can throw a java.lang.SecurityException.

Wireless Messaging API

Creating and Sending Messages

A client mode connection is created by passing a string identifying a destination address to the Connector.open () method. This method returns a MessageConnection object as follows:

```
...
/*full destination address along with port number*/
String addr = “sms://+5555555500:5432”;
MessageConnection msgconn = (MessageConnection) Connector.open(addr);
…
```

MessageConnection provides methods to create and send Messages. To create a message call the message factory method newMessage(…), and to send a message use the method send().

Creating and Sending a Text Message

The following code snippet is used for creating and sending a Text message.
Methods which are useful in this process are:

getAddress() : Returns the address associated with this message..

getTimestamp():Returns the timestamp indicating when this message has been sent.

setAddress(String): Sets the address associated with this message, that is, the address returned by the getAddress()method. The address may be set to null.

```
package example.sms;

import java.io.IOException;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.wireless.messaging.*;

public class SMSSend extends MIDlet implements CommandListener {
   Command exitCommand = new Command("Exit", Command.EXIT, 2);
   Command okCommand = new Command("OK", Command.OK, 1);
   Display display;
```

```
    String smsPort;
    TextBox destinationAddressBox;
    Alert errorMessageAlert;
    Alert sendingMessageAlert;
    SMSSender sender;
    Displayable resumeScreen = null;
    public SMSSend() {
        smsPort = getAppProperty("SMS-Port");
        display = Display.getDisplay(this);
        destinationAddressBox = new TextBox("Destination Address?", null, 256,
        TextField.PHONENUMBER);
        destinationAddressBox.addCommand(exitCommand);
        destinationAddressBox.addCommand(okCommand);
        destinationAddressBox.setCommandListener(this);
        errorMessageAlert = new Alert("SMS", null, null, AlertType.ERROR);
        errorMessageAlert.setTimeout(5000);
        sendingMessageAlert = new Alert("SMS", null, null, AlertType.INFO);
        sendingMessageAlert.setTimeout(5000);
        sendingMessageAlert.setCommandListener(this);
        sender = new SMSSender(smsPort, display, destinationAddressBox,
sendingMessageAlert);
        resumeScreen = destinationAddressBox;
        display.setCurrent(resumeScreen);
    }

    public void startApp() {

    }

    public void pauseApp() {

    }
    public void destroyApp(boolean unconditional) {
    }
    public void commandAction(Command c, Displayable s) {
        try {
         if ((c == exitCommand) /*|| (c == Alert.DISMISS_COMMAND)*/) {
              destroyApp(false);
              notifyDestroyed();
           } else if (c ==  Alert.DISMISS_COMMAND) {
              display.setCurrent(resumeScreen);
           } else if (c == okCommand) {
              promptAndSend();
```

```
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private void promptAndSend() {
        String address = destinationAddressBox.getString();

        if (!SMSSend.isValidPhoneNumber(address)) {
            errorMessageAlert.setString("Invalid phone number");
            display.setCurrent(errorMessageAlert, destinationAddressBox);

            return;
        }

        String statusMessage = "Sending message to " + address + "...";
        sendingMessageAlert.setString(statusMessage);
        sender.promptAndSend("sms://" + address);
    }
    private static boolean isValidPhoneNumber(String number) {
        char[] chars = number.toCharArray();

        if (chars.length == 0) {
            return false;
        }

        int startPos = 0;

        // initial '+' is OK
        if (chars[0] == '+') {
            startPos = 1;
        }

        for (int i = startPos; i < chars.length; ++i) {
            if (!Character.isDigit(chars[i])) {
                return false;
            }
        }

        return true;
    }
}
```

MessageListener

Whenever a message arrives, the receiving of a message can be identified by a MessageListener. For this there is a need of setting a server connection and implementing a MessageLstener. When an incoming message arrives, the notifyIncomingMessage() method is called.

The below code snippet will do it.

```
…
      smsconn = (MessageConnection)Connector.open(smsConnection);
       smsconn.setMessageListener(this);
…
```

Called by the platform when an incoming message arrives to a MessageConnection where the application has registered this listener object.

This method is called once for each incoming message to the MessageConnection.

```
public void notifyIncomingMessage(MessageConnection conn)
{
…
}
```

Receiving TextMessage

```
package example.sms;

import java.io.IOException;
import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.wireless.messaging.*;

public class SMSReceive extends MIDlet implements CommandListener, Runnable,
MessageListener {
   Command exitCommand = new Command("Exit", Command.EXIT, 2);
   Command replyCommand = new Command("Reply", Command.OK, 1);
   Alert content;
```

```
Display display;
Thread thread;
String[] connections;
boolean done;
String smsPort;
MessageConnection smsconn;
Message msg;
String senderAddress;
Alert sendingMessageAlert;
SMSSender sender;
Displayable resumeScreen;

public SMSReceive() {
        smsPort = getAppProperty("SMS-Port");
        display = Display.getDisplay(this);
        content = new Alert("SMS Receive");
        content.setTimeout(Alert.FOREVER);
        content.addCommand(exitCommand);
        content.setCommandListener(this);
        content.setString("Receiving...");
        sendingMessageAlert = new Alert("SMS", null, null, AlertType.INFO);
        sendingMessageAlert.setTimeout(5000);
        sendingMessageAlert.setCommandListener(this);

        sender = new SMSSender(smsPort, display, content, sendingMessageAlert);

        resumeScreen = content;

        display.setCurrent(resumeScreen);
}

public void startApp() {
        display.setCurrent(resumeScreen);

         String smsConnection = "sms://:" + smsPort;
        if (smsconn == null) {
      try {
        smsconn = (MessageConnection)Connector.open(smsConnection);
        smsconn.setMessageListener(this);
      } catch (IOException ioe) {
        ioe.printStackTrace();
      }
    }
```

```
    connections = PushRegistry.listConnections(true);
    if ((connections == null) || (connections.length == 0)) {
        content.setString("Waiting for SMS on port " + smsPort + "...");
    }

    done = false;
    thread = new Thread(this);
    thread.start();

}

public void notifyIncomingMessage(MessageConnection conn) {
                        if (thread == null) {
        done = false;
        thread = new Thread(this);
        thread.start();
    }
}

public void run() {
         try {
        msg = smsconn.receive();

        if (msg != null) {
            senderAddress = msg.getAddress();
            content.setTitle("From: " + senderAddress);

            if (msg instanceof TextMessage) {
                content.setString(((TextMessage)msg).getPayloadText());
            } else {
                StringBuffer buf = new StringBuffer();
                byte[] data = ((BinaryMessage)msg).getPayloadData();

                for (int i = 0; i < data.length; i++) {
                    int intData = (int)data[i] & 0xFF;

                    if (intData < 0x10) {
                        buf.append("0");
                    }

                    buf.append(Integer.toHexString(intData));
                    buf.append(' ');
```

```
                }

                content.setString(buf.toString());
            }

            content.addCommand(replyCommand);
            display.setCurrent(content);
        }
    } catch (IOException e) {
        // e.printStackTrace();
    }
}

public void pauseApp() {
    done = true;
    thread = null;
    resumeScreen = display.getCurrent();
}

public void destroyApp(boolean unconditional) {
    done = true;
    thread = null;

    if (smsconn != null) {
        try {
            smsconn.close();
        } catch (IOException e) {
            // Ignore any errors on shutdown
        }
    }
}

public void commandAction(Command c, Displayable s) {
    try {
        if (c == exitCommand || s == sendingMessageAlert ) {
                                            destroyApp(false);
            notifyDestroyed();
        } else if (c == replyCommand) {
            reply();
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
```

```
    }

    private void reply() {
        // remove the leading "sms://" for displaying the destination address
        String address = senderAddress.substring(6);
        String statusMessage = "Sending message to " + address + "...";
        sendingMessageAlert.setString(statusMessage + "\n\n...'Done' to close demo");
        sender.promptAndSend(senderAddress);
    }
}
```

The method assumes the TextMessage has already been created.

Sending binary and multipart messages follows the same pattern.

Creating Multipart Message

As MessageParts are at the center of MultipartMessages, each includes its own a MIME type, a unique content ID, and the content itself.

We can construct a MessagePart from a byte array or a java.io.InputStream. Three constructors are available:

- MessagePart(byte[] contents, int offset, int length, String mimeType, String contentId, String contentLocation, String encoding) throws SizeExceededException
- MessagePart(byte[] contents, String mimeType, String contentId, String contentLocation, String encoding) throws SizeExceededException
- MessagePart(java.io.InputStream contents, String mimeType, String contentId, String contentLocation, String encoding) throws IOException, SizeExceededException

Where:

- *Contents*: refers to the actual message content.
- *mime Type* :is the MIME type, as defined in RFC2046.
- *ContentId* :is a required ID that uniquely identifies a message part, as defined in RFC2045.
- *ContentLocation*: specifies the filename for the attached message represented by the content. A null value means that no content location value is set for this message part.
- *encoding*: identifies the content's encoding scheme. A null value means that no encoding is specified for this message part.

The first two constructors enable to create the message part from a byte array, while the last constructor used to create the message part from an InputStream.

The following APIs are useful in this process.

addAddress(String, String): Adds an address to the multipart message.

addMessagePart(MessagePart): Attaches a MessagePart to the multipart message

getAddress():Returns the "from" address associated with this message

getHeader(String): Gets the content of the specific header field of the multipart message.

getMessagePart(String): returns a MessagePart from the message that matches the content-id passed as a parameter

getStartContentId():Returns the contentId of the start MessagePart

getSubject():Gets the subject of the multipart message

removeMessagePart(MessagePart): Removes a MessagePart from the multipart message

removeMessagePartId(String): Removes a MessagePart with the specific contentID from the multipart message

removeMessagePartLocation(String): Removes MessageParts with the specific content location from the multipart message

setAddress(String): Sets the "to" address associated with this message

setHeader(String, String): Sets the specific header of the multipart message

setStartContentId(String): Sets the Content-ID of the start MessagePart of a multipart related message

setSubject(String): Sets the Subject of the multipart message.

removeAddress(String, String): Removes an address from the multipart message.

The following code snippet is used for creating a MultipartMessage.

```
package example.mms;

import java.io.InputStream;
import javax.microedition.lcdui.*;
import javax.wireless.messaging.*;

public class PartsDialog implements CommandListener {
```

```
  private static final Command CMD_BACK = new Command("Back",
Command.BACK, 1);
  private static final Command CMD_NEXT = new Command("Next", Command.OK,
1);
  private static final Command CMD_OK = new Command("OK", Command.OK, 1);
  private static final Command CMD_CANCEL = new Command("Cancel",
Command.CANCEL, 1);
  private MMSSend mmsSend;
  private List typeList;
  public int counter = 0;
  public PartsDialog(MMSSend mmsSend) {
    this.mmsSend = mmsSend;

    String[] stringArray = { "Text", "Image" };

    typeList = new List("Add Part: Type", Choice.EXCLUSIVE, stringArray, null);
    typeList.addCommand(CMD_BACK);
    typeList.addCommand(CMD_NEXT);
    typeList.setCommandListener(this);
  }

  public void show() {
    mmsSend.getDisplay().setCurrent(typeList);
  }
  public void commandAction(Command c, Displayable s) {
    try {
      if (c == CMD_BACK) {
        mmsSend.show();
      } else if (c == CMD_NEXT) {
        if (typeList.getSelectedIndex() == 0) {
          mmsSend.getDisplay().setCurrent(new TextDialog());
        } else {
          mmsSend.getDisplay().setCurrent(new ImageDialog());
        }
      }
    } catch (Exception ex) {
      ex.printStackTrace();
    }
  }

  private class TextDialog extends Form implements CommandListener {
    private Displayable mainForm;
    private TextField text;
```

```
    private String mimeType = "text/plain";

    public TextDialog() {
        super("Add Text");

        text = new TextField("Text: ", null, 256, TextField.ANY);
        append(text);
        append("MIME-Type: " + mimeType);

        addCommand(CMD_OK);
        addCommand(CMD_CANCEL);
        setCommandListener(this);
    }

    public void commandAction(Command c, Displayable s) {
        try {
            if (c == CMD_OK) {
                String encoding = "UTF-8";
                byte[] contents = text.getString().getBytes(encoding);
                mmsSend.getMessage()
                        .addPart(new MessagePart(contents, 0, contents.length, mimeType,
                         "id" + counter, "contentLocation", encoding));
                counter++;
                mmsSend.show();
            } else if (c == CMD_CANCEL) {
                mmsSend.show();
            }
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
}
private class ImageDialog extends Form implements CommandListener {
    private Displayable mainForm;
    private ChoiceGroup cg;
    private String mimeType = "image/png";
    private String[] resouces = { "/Java.png", "/Duke.png" };
    private String[] imagesNames = { "Java", "Duke" };

    public ImageDialog() {
        super("Add Image");
        cg = new ChoiceGroup("Select Image", Choice.EXCLUSIVE, imagesNames, null);
        append(cg);
```

```
            append("MIME-Type: " + mimeType);
            addCommand(CMD_OK);
            addCommand(CMD_CANCEL);
            setCommandListener(this);
        }

        public void commandAction(Command c, Displayable s) {
            try {
                if (c == CMD_OK) {
                    int index = cg.getSelectedIndex();
                    String resouce = resouces[index];
                    InputStream is = getClass().getResourceAsStream(resouce);
                    byte[] contents = new byte[is.available()];
                    is.read(contents);

                    String contentLocation = imagesNames[index];
                    mmsSend.getMessage()
                            .addPart(new MessagePart(contents, 0, contents.length, mimeType,
                             "id" + counter, contentLocation, null));
                    counter++;
                    mmsSend.show();
                } else if (c == CMD_CANCEL) {
                    mmsSend.show();
                }
            } catch (Exception ex) {
                ex.printStackTrace();
            }
        }
    }
}
```

Sending Multipart Message

The below mentioned code snippet is useful for sending multipart message:

```
package example.mms;

import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.wireless.messaging.*;
```

```
public class MMSSend extends MIDlet implements CommandListener {
   private static Command CMD_EXIT = new Command("Exit", Command.EXIT, 2);
   private static Command CMD_SEND = new Command("Send", Command.ITEM, 1);
   private static Command CMD_ADD_PART = new Command("Add Part",
Command.ITEM, 1);
   private Display display;
   private String appID;
   private TextField subjectField;
   private TextField destinationField;
   private StringItem partsLabel;
   private Alert errorMessageAlert;
   private Alert sendingMessageAlert;
   private Displayable resumeScreen = null;
   private MMSMessage message;
   private PartsDialog partsDialog;
   public MMSSend() {
      appID = getAppProperty("MMS-ApplicationID");
      display = Display.getDisplay(this);
      Form mainForm = new Form("New MMS");
      subjectField = new TextField("Subject:", null, 256, TextField.ANY);
      mainForm.append(subjectField);
      destinationField = new TextField("Destination Address: ", "mms://", 256,
TextField.ANY);
      mainForm.append(destinationField);
      partsLabel = new StringItem("Parts:", "0");
      mainForm.append(partsLabel);
      mainForm.addCommand(CMD_EXIT);
      mainForm.addCommand(CMD_SEND);
      mainForm.addCommand(CMD_ADD_PART);
      mainForm.setCommandListener(this);
      errorMessageAlert = new Alert("MMS", null, null, AlertType.ERROR);
      errorMessageAlert.setTimeout(5000);
      sendingMessageAlert = new Alert("MMS", null, null, AlertType.INFO);
      sendingMessageAlert.setTimeout(5000);
      sendingMessageAlert.setCommandListener(this);

      resumeScreen = mainForm;

      message = new MMSMessage();
   }
   public void startApp() {
      display.setCurrent(resumeScreen);
   }
```

```
public void pauseApp() {
    resumeScreen = display.getCurrent();
}
public void destroyApp(boolean unconditional) {
}
public void commandAction(Command c, Displayable s) {
    try {
        if ((c == CMD_EXIT) ) {
            destroyApp(false);
            notifyDestroyed();
        } else if (c == Alert.DISMISS_COMMAND) {
             display.setCurrent(resumeScreen);
        } else if (c == CMD_ADD_PART) {
            if (partsDialog == null) {
                partsDialog = new PartsDialog(this);
            }
            partsDialog.show();
        } else if (c == CMD_SEND) {
            promptAndSend();
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
void show() {
    partsLabel.setText(Integer.toString(partsDialog.counter));
    display.setCurrent(resumeScreen);
}
Display getDisplay() {
    return display;
}
MMSMessage getMessage() {
    return message;
}
private void promptAndSend() {
    try {
        String address = destinationField.getString();
        message.setSubject(subjectField.getString());
        message.setDestination(address);

        String statusMessage = "Sending message to " + address + "...";
        sendingMessageAlert.setString(statusMessage);
```

```
            new SenderThread(message, appID).start();
        } catch (IllegalArgumentException iae) {
            errorMessageAlert.setString(iae.getMessage());
            display.setCurrent(errorMessageAlert);
        }
    }
}
```

Receiving Multipart Message

The following code snippet illustrates how to create two message parts, one text/plain and the other image/png.

```
import java.io.IOException;

import javax.microedition.io.*;
import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;
import javax.wireless.messaging.*;
public class MMSReceive extends MIDlet implements CommandListener, Runnable,
MessageListener {
    private static final Command CMD_EXIT = new Command("Exit", Command.EXIT,
2);
    private Form content;
    private Display display;
    private Thread thread;
    private String[] connections;
    private boolean done;
    private String appID;
    private MessageConnection mmsconn;
    private Message msg;
    private String senderAddress;
    private Alert sendingMessageAlert;
    private Displayable resumeScreen;
    private String subject;
    private String contents;
    public MMSReceive() {
        appID = getAppProperty("MMS-ApplicationID");
        display = Display.getDisplay(this);
        content = new Form("MMS Receive");
        content.addCommand(CMD_EXIT);
        content.setCommandListener(this);
```

```
    content.append("Receiving...");
    sendingMessageAlert = new Alert("MMS", null, null, AlertType.INFO);
    sendingMessageAlert.setTimeout(5000);
    sendingMessageAlert.setCommandListener(this);
    resumeScreen = content;
}
public void startApp() {
          display.setCurrent(resumeScreen);
    String mmsConnection = "mms://:" + appID;
    if (mmsconn == null) {
       try {
          mmsconn = (MessageConnection)Connector.open(mmsConnection);
          mmsconn.setMessageListener(this);
       } catch (IOException ioe) {
          ioe.printStackTrace();
       }
    }
    connections = PushRegistry.listConnections(true);

    if ((connections == null) || (connections.length == 0)) {
       content.deleteAll();
       content.append("Waiting for MMS on applicationID " + appID + "...");
    }
    done = false;
    thread = new Thread(this);
    thread.start();
}

public void notifyIncomingMessage(MessageConnection conn) {
    if ((thread == null) && !done) {
       thread = new Thread(this);
       thread.start();
    }
}
public void run() {
    try {
       msg = mmsconn.receive();
       if (msg != null) {
          senderAddress = msg.getAddress();
          content.deleteAll();
          String titleStr = senderAddress.substring(6);
          int colonPos = titleStr.indexOf(":");
          if (colonPos != -1) {
```

```
            titleStr = titleStr.substring(0, colonPos);
        }
        content.setTitle("From: " + titleStr);
        if (msg instanceof MultipartMessage) {
            MultipartMessage mpm = (MultipartMessage)msg;
            StringBuffer buff = new StringBuffer("Subject: ");
            buff.append((subject = mpm.getSubject()));
            if (mpm.getTimestamp() != null) {
                buff.append("\nDate: ");
                buff.append(mpm.getTimestamp().toString());
            }
            buff.append("\nContent:");
           StringItem messageItem = new StringItem("Message", buff.toString());
            messageItem.setLayout(Item.LAYOUT_NEWLINE_AFTER);
            content.append(messageItem);
            MessagePart[] parts = mpm.getMessageParts();
            if (parts != null) {
                for (int i = 0; i < parts.length; i++) {
                    buff = new StringBuffer();
                    MessagePart mp = parts[i];
                    buff.append("Content-Type: ");
                    String mimeType = mp.getMIMEType();
                    buff.append(mimeType);
                    String contentLocation = mp.getContentLocation();
                    buff.append("\nContent:\n");
                    byte[] ba = mp.getContent();

                    try {
                        Image image = Image.createImage(ba, 0, ba.length);
                        content.append(buff.toString());
                        ImageItem imageItem =
                            new ImageItem(contentLocation, image,
                                Item.LAYOUT_NEWLINE_AFTER, contentLocation);
                        content.append(imageItem);
                    } catch (IllegalArgumentException iae) {
                        buff.append(new String(ba));

                        StringItem stringItem = new StringItem("Part", buff.toString());
                        stringItem.setLayout(Item.LAYOUT_NEWLINE_AFTER);
                        content.append(stringItem);
                    }
                }
            }
```

```
            }

            display.setCurrent(content);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void pauseApp() {
    done = true;
    thread = null;
    resumeScreen = display.getCurrent();
}
public void destroyApp(boolean unconditional) {
    done = true;
    thread = null;

    if (mmsconn != null) {
        try {
            mmsconn.close();
        } catch (IOException e) {
            // Ignore any errors on shutdown
        }
    }
}
public void commandAction(Command c, Displayable s) {
    try {
        if ((c == CMD_EXIT)) {
            destroyApp(false);
            notifyDestroyed();
        } else if (c == Alert.DISMISS_COMMAND) {
            startApp();
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
}
```

There is a need to check the validity of the phone number. The following code snippet will do the required one.

```
package example.mms;

import java.util.Vector;
import javax.wireless.messaging.*;

public class MMSMessage {
   private String destination;
   private Vector parts = new Vector();
   private String subject;
   private static boolean isValidPhoneNumber(String address) {
      String protocol = "mms://";
      if (!address.startsWith(protocol)) {
         return false;
      }
      String number = address.substring(protocol.length());
      char[] chars = number.toCharArray();
      if (chars.length == 0) {
         return false;
      }
      int startPos = 0;
      if (chars[0] == '+') {
         startPos = 1;
      }
      for (int i = startPos; i < chars.length; ++i) {
         if (!Character.isDigit(chars[i])) {
            return false;
         }
      }
      return true;
   }

   public String getSubject() {
      return subject;
   }

   public void setSubject(String subject) {
      this.subject = subject;
   }

   public String getDestination() {
      return destination;
   }
```

```
    public void setDestination(String destination) {
        if (!isValidPhoneNumber(destination)) {
            throw new IllegalArgumentException("Invalid phone number");
        }

        this.destination = destination;
    }

    public MessagePart[] getParts() {
        MessagePart[] partsArray = new MessagePart[parts.size()];
        parts.copyInto(partsArray);

        return partsArray;
    }

    public void addPart(MessagePart part) {
        parts.addElement(part);
    }
}
```

Send the message by calling on a separate thread so there won't be any contention for the display.

Use the following code snippet for this.

```
package example.mms;
import java.io.IOException;
import javax.microedition.io.*;
import javax.wireless.messaging.*;
public class SenderThread extends Thread {
    private MMSMessage message;
    private String appID;

    public SenderThread(MMSMessage message, String appID) {
        this.message = message;
        this.appID = appID;
    }
    public void run() {
        String address = message.getDestination() + ":" + appID;
        MessageConnection mmsconn = null;

        try {
            mmsconn = (MessageConnection)Connector.open(address);
```

```
        MultipartMessage mmmessage =

(MultipartMessage)mmsconn.newMessage(MessageConnection.MULTIPART_MESSAG
E);
        mmmessage.setAddress(address);

        MessagePart[] parts = message.getParts();

        for (int i = 0; i < parts.length; i++) {
           mmmessage.addMessagePart(parts[i]);
        }

        mmmessage.setSubject(message.getSubject());
        mmsconn.send(mmmessage);
     } catch (Exception e) {
        e.printStackTrace();
     }

     if (mmsconn != null) {
        try {
           mmsconn.close();
        } catch (IOException ioe) {
           ioe.printStackTrace();
        }
     }
  }
}
```