Wireless Messaging API (WMA)

Version 0.9, Draft

API GUIDE

COPYRIGHT

Trademarks and Service Marks

About This Document

This document will cover in brief description about WMA (Wireless Messaging API) and demonstrates building sample WMA application for Samsung mobiles. It includes the overview of WMA Architecture and APIs, followed by a Sample code snippet on how to send and receive SMS.

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
05/02/09	0.9	Draft

References:

1. WMA 1.1 Specification:

   http://jcp.org/en/jsr/detail?id=120

2. WMA

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

Abbreviations:

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

Table of Contents

Introduction 5
Overview 5
Cell Broadcast Service (CBS) 5
Short Messaging Service (SMS) 5
API Description 7
Interfaces 7
Sending and Receiving messages using WMA 8
Sending a text message 8
Receiving a text message 9
Sample code snippet to send a text message 9
Sample code snippet to receive a Text Message 12

Table of Figures

Figure 1: Generic Connection Framework. 6
Figure 2: Wireless Message API Components 7

Introduction

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

Overview

The following messaging protocols are explained in this document:

- Cell Broadcast Service (CBS)
- Short Messaging Service (SMS)

Cell Broadcast Service (CBS)

- With the help of GSM cell broadcast service, messages can be sent to every Mobile Station (MS). Examples: Mobile phones and Fax machines.
- Periodically the cell broadcast messages are repeated, so even entering the cell after the first transmission message can be received to an MS.
- Binary data and ASCII are the two ways to send the data. In ASCII, length of text is up to 15 pages with 93 characters per page; the text set only provides support for ASCII messages.

Short Messaging Service (SMS)

- Short Messaging Service is a communication protocol for transmission of short text/binary message to and from mobile phone originally defined as part of the GSM series of standards.
- Maximum length of each message is 160 characters.

This document covers the Wireless Messaging API in brief. For more information, refer to the specification defined by Java Community Process(JCP).. 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*.

As shown in Figure 1 the message sending and receiving functionality is implemented by a *MessageConnection* (derived from Connection interface of Generic Connection Framework (GCF)). Figure 1 describes the relation of WMA with GCF.

The methods for sending and receiving messages can throw a *java.lang.SecurityException*, if the application does not have the permission to perform these operations.

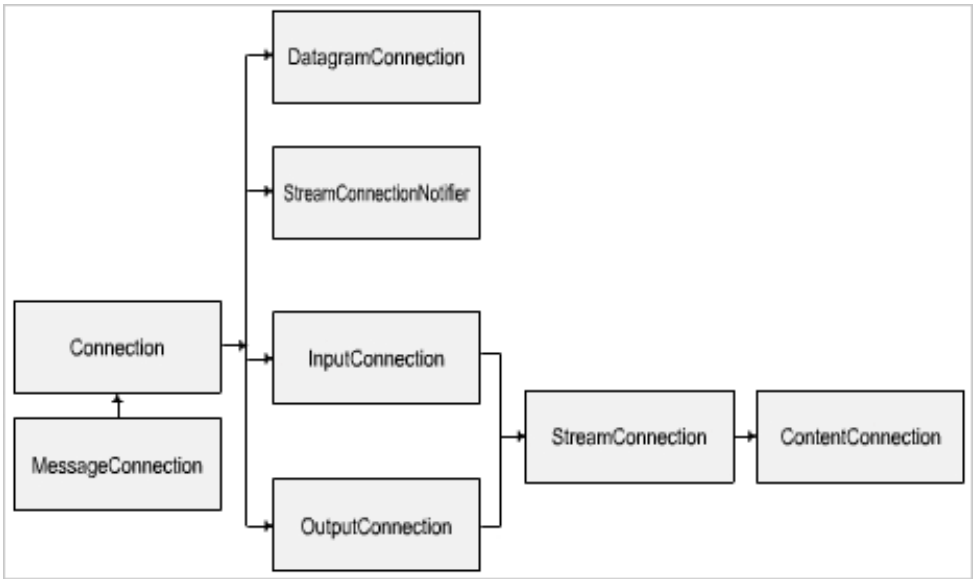

Figure 1: Generic Connection Framework.

API Description:

WMA API consists of interfaces in a single package, *javax.wireless.messaging*, which defines all the APIs required for sending and receiving wireless text and binary messages. Given below are WMA specific interfaces:

Message
BinaryMessage
TestMessage
MessageConnection
MessageListener

Figure 2 shows the Wireless Message API Components:

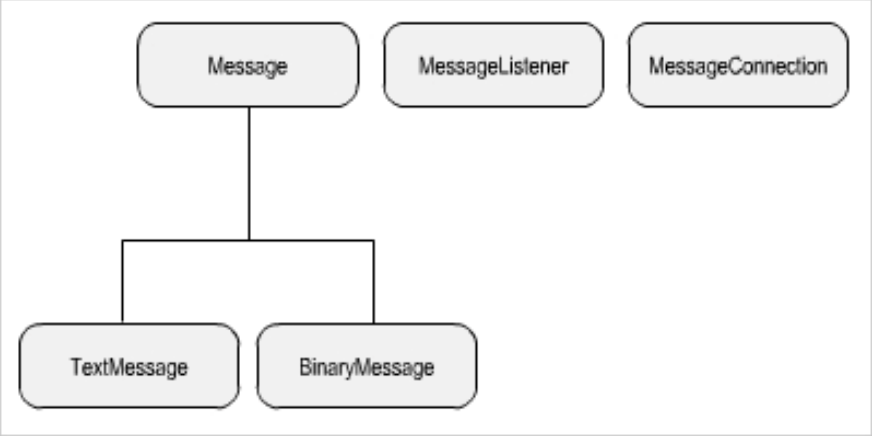

Figure 2: Wireless Message API Components

Interfaces

Message:

This is the base Interface. *TextMessage* and *BinaryMessage* are derived from Message interface. It acts as a container holding address, payload for the message. Message has methods for addresses and timestamps. These methods are *getAddress()*, *getTimeStamp()*, *setAddress()*.

BinaryMessage:

This is *subinterface* of Message. This interface is used for sending and receiving binary message. Methods to set and get the binary payload data are *getPayloadData()*, *setPayloadData()*.

TextMessage:

This is *subinterface* of Message. This interface is used for sending and receiving text message. Methods to set and get the text payload data are *getPayloadText()*, *setPayloadText().*

MessageConnection:

This is *subinterface* of the GCF Connection, which contains a factory method for creating new *Message* objects. *MessageConnection* contains method *numberOfSegments()* which returns the number of segments of the underlying protocol that is needed to send a specified *Message*. Other methods available in this interface are *newMessage(), receive(), send(), setMessageListener().*

MessageListener:

This is a listener interface, which listens to the notification of incoming messages through *notifyIncomingMessage()* method.

Sending and Receiving messages using WMA

Code walkthrough for sending and receiving messages using Wireless Messaging API:

Sending a text message

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*/
String addr = "sms://+5555555500:5432";
MessageConnection msgconn = (MessageConnection) Connector.open(addr);
```

The address part contains telephone number with country code and port number, which is application specific. The 3rd Generation Partnership Project (3GPP) specification for SMS specifies the port numbers 16000-16999 as available for applications. Some of the handsets reserve ports (restricted port) for system. System reserved ports may not be used by the Java ME applications. Using of non reserved port within the specified range is recommended for developing Java ME applications.

Then pass *MessageConnection.TEXT_MESSAGE* constant in *newMessage()* method.
This method returns *TextMessage* object as follows:

```
TextMessage txtmsg =
   (TextMessage)msgconn.newMessage(MessageConnection.TEXT_MESSAGE);
```

Provides the payload text using *Textmessage* object.

```
txtmsg.setPayloadText("Samsung Mobile Innovator");
```

Uses *MessageConnection* object to send the text.

```
msgconn.send(msg);
```

Receiving a text message

Message recieving in Java ME is mainly associated with a port. To recieve message, a server mode connection is created by passing a string that identifies an end point (Protocol identifier, for example, a port number) to the *Connector.open()* method. This method returns a *MessageConnection* object as follows:

```
/* address part*/
String addr = "sms://:5432";
MessageConnection msgconn = (MessageConnection) Connector.open(addr);
```

The format of the URL connection string that identifies the address is specific to the messaging protocol used.

To receive a message use *receive()* method which returns a Message object.

```
msg = conn.receive();
```

Check whether the message received is *Textmessage* and get the payload text.

```
if (msg instanceof TextMessage) {
    TextMessage tmsg = (TextMessage)msg;
    String receivedText = tmsg.getPayloadText();
}
```

Sample code snippet to send a text message

The sample code given below shows how to send a text message.

This code assumes that *portNo* attribute is added in JAD. Ex: *portNo: 5000*

Class: SendSMS

```
import javax.microedition.io.Connector;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.lcdui.Form;
import javax.microedition.lcdui.TextBox;
```

```
import javax.microedition.lcdui.TextField;
import javax.microedition.midlet.MIDlet;
import javax.wireless.messaging.MessageConnection;
import javax.wireless.messaging.TextMessage;

public class SendSMS extends MIDlet implements CommandListener, Runnable {

   private static final String EXIT = "Exit";
   private static final String OK = "Ok";
   private static final String BACK = "Back";
   private static final String SEND = "Send";
   private Form form;
   private TextField textfield;
   private TextBox textbox;
   private Display display;
   private Thread thread;

   /*this command is used to exit from the current application*/
   Command cmdExit = new Command(EXIT, Command.EXIT, 2);
   /*command for acceptance to display next screen*/
   Command cmdOk = new Command(OK, Command.OK, 1);
   /*command which allows to move to previous screen*/
   Command cmdBack = new Command(BACK, Command.BACK, 2);
   /*command to send the entered message to destination*/
   Command cmdSend = new Command(SEND, Command.OK, 1);
   private String getPort = null;

   public SendSMS() {
      /*gets the port number from JAD*/
      getPort = this.getAppProperty("portNo");
      display = Display.getDisplay(this);
      form = new Form("Enter Destination address");
      textfield = new TextField("Enter Destination address:", "", 20,
            TextField.PHONENUMBER);
      form.append(textfield);
      form.addCommand(cmdExit);
      form.addCommand(cmdOk);
      form.setCommandListener(this);
   }

   public void startApp() {
      display.setCurrent(form);
   }

   public void pauseApp() {
      /*all interupts can be handled here*/
   }
```

```
public void destroyApp(boolean unconditional) {
}

public void commandAction(Command cmd, Displayable disp) {
   if (cmd == cmdExit && disp == form) {
      exitMidlet();
   } else if (cmd == cmdOk && disp == form) {
      textbox = new TextBox("Enter Message", null, 256, TextField.ANY);
      textbox.addCommand(cmdBack);
      textbox.addCommand(cmdSend);
      textbox.setCommandListener(this);
      display.setCurrent(textbox);
   } else if (cmd == cmdBack && disp == textbox) {
      display.setCurrent(form);
   } else if (cmd == cmdSend && disp == textbox) {
      sendSMS();
   }
}

public void sendSMS() {
   /*recomeneded to start the players in a separate thread*/
   thread = new Thread(this);
   thread.start();
}

public void run() {
   MessageConnection msgconn = null;
   try {
      /*full destination address*/
      String address = "sms://" + textfield.getString() + ":" + getPort;
      /*to open message connection*/
      msgconn = (MessageConnection) Connector.open(address);
      /*to send text message*/
      TextMessage textmsg = (TextMessage)
            msgconn.newMessage(MessageConnection.TEXT_MESSAGE);
      /*pay load text passed here*/
      textmsg.setPayloadText(textbox.getString());
      /*send the message*/
      msgconn.send(textmsg);
   } catch (Exception exe) {
      exe.printStackTrace();
   } finally {
      try {
         msgconn.close();
      } catch (Exception io) {
         io.printStackTrace();
```

```
        }
        exitMidlet();
      }
    }

    public void exitMidlet() {
      destroyApp(false);
      notifyDestroyed();
    }
}
```

Sample code snippet to receive a Text Message

The sample code given below shows how to receive a text message:

This code assumes that *portNo* attribute is added in JAD. Ex: *portNo: 5000*

Class: ReceiveSMS

```
import javax.microedition.io.Connector;
import javax.microedition.lcdui.Alert;
import javax.microedition.lcdui.Command;
import javax.microedition.lcdui.CommandListener;
import javax.microedition.lcdui.Display;
import javax.microedition.lcdui.Displayable;
import javax.microedition.midlet.MIDlet;
import javax.wireless.messaging.BinaryMessage;
import javax.wireless.messaging.Message;
import javax.wireless.messaging.MessageConnection;
import javax.wireless.messaging.MessageListener;
import javax.wireless.messaging.TextMessage;

public class ReceiveSMS extends MIDlet implements CommandListener,
      MessageListener, Runnable {

    private static final String EXIT = "Exit";
    private Display display;
    private Thread thread;
    private MessageConnection msgconn;
    private Message msg;
    private String senderAddress;
    private Alert recMess;

    /*this command is used to exit from the current application*/
    Command cmdExit = new Command(EXIT, Command.EXIT, 2);
    private String getPort = null;
```

```
public ReceiveSMS() {
    /*gets the port number from JAD*/
    getPort = this.getAppProperty("portNo");
    display = Display.getDisplay(this);
    recMess = new Alert("Message");
    recMess.setString("Waiting for SMS");
    recMess.setTimeout(Alert.FOREVER);
    recMess.addCommand(cmdExit);
    recMess.setCommandListener(this);
}

public void startApp() {
    display.setCurrent(recMess);
    try {
        /*port on which to receive sms*/
        String address = "sms://:" + getPort;
        /*open connection*/
        msgconn = (MessageConnection) Connector.open(address);
        /*set listening mode*/
        msgconn.setMessageListener(this);
        /*need to start the players in a separate thread to
        avoid dead lock*/
        thread = new Thread(this);
        thread.start();
    } catch (Exception exe) {
        exe.printStackTrace();
    }

}

public void pauseApp() {
    thread = null;
/*all interupts can be handled here*/
}

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

    if (msgconn != null) {
        try {
            msgconn.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
  public void commandAction(Command cmd, Displayable disp) {
    if (cmd == cmdExit && disp == recMess) {
      exitMidlet();
    }
  }

  public void exitMidlet() {
    destroyApp(true);
    notifyDestroyed();
  }

  public void notifyIncomingMessage(MessageConnection msgconn) {
    thread = new Thread(this);
    thread.start();
  }

  public void run() {
    try {
      /*gets message object*/
      msg = msgconn.receive();
      if (msg != null) {
        /*gets address of the message received*/
        senderAddress = msg.getAddress();
        recMess.setTitle("From: " + senderAddress);
        /*check for the type of message received*/
        if (msg instanceof TextMessage) {
          recMess.setString("Recieved Messsage: \n" + ((TextMessage) msg).getPayloadText());
        } else {
          /*get the binary data of the message*/
          byte[] data = ((BinaryMessage) msg).getPayloadData();

        /*wite here code to convert binary to text message*/

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