



Using the WMA Test Server for MMS Messaging

October 1, 2006

Technical Article

Using the WMA Test Server for MMS Messaging

By
MOTODEV Staff

The Motorola [J2ME SDK for Linux Products](#) supports JSR 205 for sending and receiving Short Message Service messages (SMS), Cell Broadcast Service messages (CBS) as well as Multimedia Messaging Service messages (MMS). This SDK includes a complete messaging simulation test environment for sending and receiving these messages between programs running in an emulator and a messaging server as well as simulating the GSM network that carries these messages. In this article we will examine the usage of the WMA Test Server for sending and receiving MMS messages and will refer to sections of the sample MIDlet source code provided to further explain the WMA Test Server resources. We will avoid focusing on the WMA 2.0 MMS APIs themselves in this article and will be instead taking a look at the WMA Test Server and its usage. The [sample code](#) that accompanies this article has been tested using the JSR 205 supported emulated handsets in the J2ME SDK for Linux Products. The sample can be downloaded from <http://developer.motorola.com/?path=1.2.888/>. Let's begin.

WMA Test Server Overview

The WMA Test Server is a program that you run on your PC to simulate a messaging server and the network that carries the messages to and from handsets. As the developer, you have control over uploading messages to the test server to simulate sending and also downloading messages from the test server to simulate receiving on your MIDlets. You can simulate the messaging process between your MIDlet and the server, but not between MIDlets running in separate emulators on your development PC. The server is an executable and is located in the <SDK_Home>/AplixTestWMAserver directory of your SDK. It can be correctly started from a DOS window by navigating to the directory and issuing the following command:

```
TestWmaServer -server_port 20001 -client_port 20000
```

This will start the server running in its own DOS window as shown in Figure 1.

p6ipqrKztLW2t7i5usLDxMXGx8jJytLT1NXW19jZ2uHi4+Ti5ufo6erx8vP09fb3+Pn6/8QAHwEA
AwEBAQEBAQEBAQAAAAAAAECAwQFBgcICQoL/8QAtREAAgECBAQDBAcFBAQAAQJ3AAECAxEEBSEx
BhJBUQdhcRMiMoEIFEKRobHBCSMzUvAVYnLRChYkNOEl8RcYGRomJygpKjU2Nzg5OkNERUZHSEIK
U1RVVldYWVpjZGVmZ2hpanN0dXZ3eHl6goOEhYaHiImKkpOUlZaXmJmaoqOkpaanqKmqsR00tba3
uLm6wsPExcbHyMnK0tPU1dbX2Nna4uPk5ebn6Onq8vP09fb3+Pn6/9oADAMBAAIRAxEAPwDs6KKz
tW1m00WK2lvCVjnuEtw/ACIs/MxJGFGDk19JKSSuz8TpUp1ZqEFds0aKzV1m0fxBJoyEtdR2/wBo
crgqo3AbTzkNyDjHQg1pURknsVUo1KTXMrXV/kwooqt9utv7R/s/zf8AS/K8/wAvafuZ25zjHXjG
c020tyIU5Tvyq9izRRRTMgooooAKKKKACiiigAooooAK4T4rwy3HhW1hgjaSV76MKiDLMSrgAADT
Xd1VvLC2vvi+0xb/ACJini+Yja652twecZPB4rKtDng4rqehluKjhsTGtNXSZ5N4Yv8AW/EPIXWN
Ut9tlqV1ppeBki+RijouFDkjDbCuScA59K7bRPEz+KNXtX0yUJY21t5l8uzOZX4WLLAH5cM25eDj
Falr4a03TgjabG9ILHCbeOWNtxSMYeYVafcpypXJBPNXNN0230u0FvbA4LF3kdtzyufvOx/iY9z/
AEAF9KjUhZN+v8AX5ntY/NMJiluUlapWjdbK1nfe+IrdnqXawv7Ruv+E6/szzP9E/sz7RswPv8A
m7c5xnpXjOK3aq/YLb+0v7Q8r/S/K8jzNx+5u3bcZx15zjNdM03ax4WFqU6fN7RXumI69zze+8S+
IV1i7WPVilv5+oRxxLbx5QQRyZnFTkHIB74HXnjZvNfv5dNlura9uGuV0yO7eCygi8u2YpuzK0m
SdxOQq4baOh6noJPC+jSzNK9nmRnmcnzX6zLskP3v4lGPbtim3HhTRbmFoZLI+U8SROiTSisipwg
cKw3FexOTwOa5FQRK/vfiz6F5nlslH91a2/ux1/H+vvMbVvEI1FaeGpTdLarqEZluBCEh/dBvka
X5FUMwzuOemM81BhROtw6boNk959sv8AVHkeW5tfJfYseCUTO1CccFiTg7sbuMdD/wAI1awTW81j
cXVILBb/AGVXicOTFnIU+YG4B6YxgdcAAHReHNPjsZLWRJJxJcm7eWRtrmYnO8Mu3aeAPlxx9Tmv
Z1m9/wAft+tjL67l8acUoJ2b+yr9bX8lddbW0DQ5dXf7VFqdvql4NtPIYw8iHJw6xkqCvTlxkE
cA5rYqLZaZaae0zW8ZEK7BpZJJGkdyBgZdyWIA6DOBz61drqppqNmeJi6tOpVcoKyfZW6drv8woo
oqzhCiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAooooAKKKKACiiigAoo
oAKKKKACiiigAooooA/9k=
-----_6011093==_

If you do elect to create the text file to send from the WMA Test Server with email software, the steps are as follows for Outlook Express.

1. Start Outlook Express or a similar mail client.
2. Select "Create Mail"
3. Enter your addressees, subject and message body and any media attachments as necessary.
4. Select File -> Save As and save the message as .eml format on your PC.
5. Perform any necessary editing of the .eml file so that it conforms to our template example above and save as text.

To receive MMS on our MIDlet from the WMA Test Server, we need to connect in server mode since only server mode allows receiving messages. For MMS messaging, the URL connection parameter takes this format:

(MessageConnection)Connector.open("mms://:<Application-ID>");

The <Application-ID> is the full class path of the MMS receiving MIDlet running in your emulator. In our example we use the MIDlet.getAppProperty method to retrieve application properties from the JAD file. In our JAD file we have specified the MMS-ApplicationID as the full package and class name of our MMS receiving MIDlet as shown here.

MMSTester JAD file
MIDlet-1: MMSTester, .com.mot.mmstester.MMSTester
MIDlet-Jar-Size: 6800
MIDlet-Jar-URL: mmstester.jar
MIDlet-Name: MMSTester
MIDlet-Vendor: MotoDev

MIDlet-Version: 1.0

MMS-ApplicationID: com.mot.mmstester.MMSTester

The source code that creates this connection is included here.

Note: The connection URLs used in this MIDlet are for connecting to the WMA Test Server for testing purposes and will not allow sending of MMS message to other handsets. For URL connections for messaging other handsets please follow the recommendation in "[Introduction of MMS in J2ME](#)" from September's MOTODEV Newsletter.

Code sample 1

```
***
private String appID; // declare variable to hold Application-ID
***
appID = getAppProperty( "MMS-ApplicationID" ); // Retrieve Application-ID
from JAD file
***
String mmsConnection = "mms://:" + appID;
if ( mmsconn == null ) {
    try {
        mmsconn = ( MessageConnection ) Connector.open( mmsConnection );
    } catch ( IOException ioe ) {
        dbg( "Connector.open: " + ioe.toString() );
        mainScreen.append( ioe.toString() );
    } // end catch
} // end if
```

Once this connection is established we can wait to receive messages from the WMA Test Server. This is best done is a separate thread so our MIDlet's user interface will remain responsive while we are waiting for messages.

Code sample 2

```
public void run() {
    (1) while( !done ) {
        try {
            msg = mmsconn.receive();
            if ( msg != null ) {
                mainScreen.deleteAll();
                senderAddress = msg.getAddress();
                if( senderAddress != null ) {
                    mainScreen.append( "Message from: " + senderAddress );
                }
                (2)if ( msg instanceof MultipartMessage ) {
                    MultipartMessage mpm = (MultipartMessage)msg;
                    // Subject
                    mainScreen.append( "Subject: " + (3) mpm.getSubject() );
                    // Date
                    mainScreen.append( "Date: " + (4) mpm.getTimestamp().toString() );

                    // Display something for each part
```

```

(5) MessagePart[] parts = mpm.getMessageParts();
if ( parts != null ) {
    for ( int i = 0; i < parts.length; i++ ) {
        MessagePart mp = parts[i];
        mainScreen.append( "Content-Type: " + mp.getMIMEType() );
        byte[] ba = mp.getContent();
        String contentLocation = mp.getContentLocation();
        // Depending on the mime type do something
        if(      (6)      mp.getMIMEType().trim().substring(      0,
5 ).equals( "image" ) ) {
            try {
                Image image = Image.createImage( ba, 0, ba.length );
                ImageItem imageItem = new ImageItem( contentLocation, image,
Item.LAYOUT_NEWLINE_AFTER, contentLocation );
                mainScreen.append( imageItem );
            } catch (      IllegalArgumentException      iae      )
{ dbg( "IllegalArgumentException createImage" ); }
        } else if(      mp.getMIMEType().trim().substring(      0,
5 ).equals( "audio" ) ) {
            // Do something with audio file, play etc..
            mainScreen.append( "Audio file is attached" );
        } else {
            // We'll assume this is a String then. If not whatever it is will
be printed to screen as text.
            mainScreen.append( new String( ba ) );
        }
    } // end for
} // end if
// Set the boolean flag to true here
(7) done = true;
} // end if
display.setCurrent( mainScreen );
}
} catch (IOException e) {
    e.printStackTrace();
}
} // end while loop
Timer timer = new Timer();
TimerTask task = new ThreadTask();
// Restart in thread in 3 seconds
timer.schedule( task, 3000 );
} // end run method

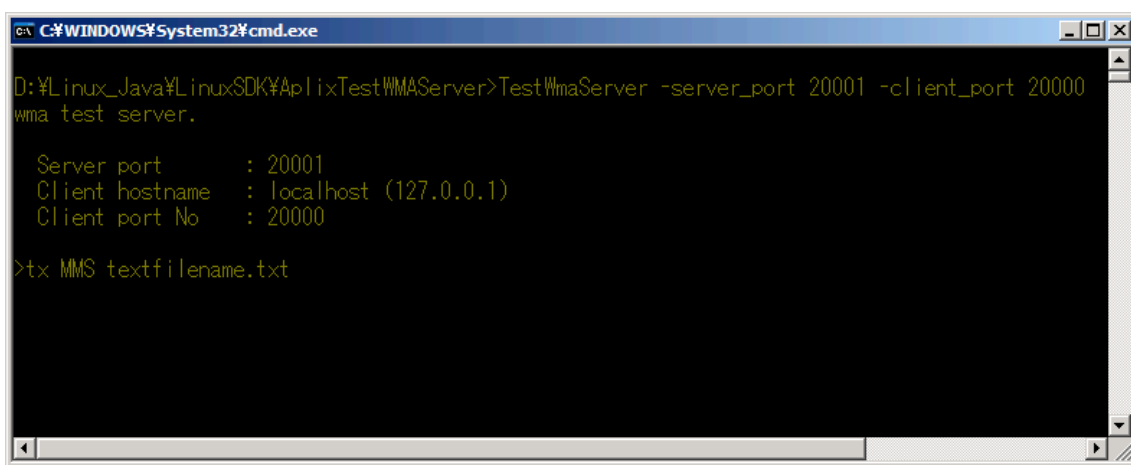
```

The Boolean value done is declared as false and is only set to true after message reception. So we are continuously looping through our while loop waiting for messages from the connection we established earlier (1). When a Message object is received, we check to see that it is a Multipart message (2) and if so, we proceed to retrieve the message subject with the (3) Message.getSubject method, the date of the message with the (4) Message.getTimestamp method. The body and any media attached to our message can be retrieved with the (5)

`Message.getMessageParts` method. Depending on the MIME type returned by the (6) `Message.getMIMEType` method, we can choose to display or discard the `MessagePart` as we wish. Finally, our Boolean value is set to `true` since we have received our message (7).

Now that the MIDlet has established the server connection to the appropriate port and our MIDlet is waiting to receive messages, let's take a look at how we can send the MMS message from the WMA Test Server command window. From our WMA Test Server command window, the following command where `textfilename.txt` is the name of the MMS message text file you prepared will send the MMS message to our waiting MIDlet.

```
>tx MMS textfilename.txt
```



```
C:\WINDOWS\System32\cmd.exe
D:\Linux_Java\LinuxSDK\ApI\ixTest\WMA Server>TestWmaServer -server_port 20001 -client_port 20000
wma test server.

Server port      : 20001
Client hostname  : localhost (127.0.0.1)
Client port No   : 20000

>tx MMS textfilename.txt
```

Figure 2: Sending the MMS message from the command line

Sending messages to the WMA Server

Sending an MMS message from your MIDlet running in an emulator on your PC to the WMA Test Server running on the same PC is relatively straightforward. The MMS message can be constructed by the MIDlet and simply sent to the WMA Test Server listening on the correct Port.

From our command issued to start the server and as Figure 1 demonstrates, our server is listening on Port 20001 for incoming messages. So, if our MMS sending MIDlet wishes to send to the server we first need to open a connection to this port. We can use the `Connector.open` factory method with our connection URL to return a `MessageConnection` object. The connection and sending of the message is processor heavy and should be carried out in a separate thread as shown in our sample's run method here.

Code sample 3

```
String addr = "mms://:" + 20001;
*****
public void run() {
    try {
        MessageConnection conn = ( MessageConnection ) Connector.open( addr );
        MultipartMessage mmmmessage =
```

```

    (MultipartMessage)conn.newMessage( MessageConnection.MULTIPART_MESSAGE );
    (1)mmessage.setAddress( addr );
    MessagePart part = getBody();
    (2)mmessage.addMessagePart( part );
    (3)mmessage.setSubject( subjectTextField.getString() );
    conn.send( mmessage );
} catch( Exception e ) { out( e.toString() ); }
} // end run
*****

```

Our MIDlet uses the MultipartMessage API to build the MMS message before calling the MessageConnection.send method to send our MMS message to the WMA Test Server. The (1) MultipartMessage.setAddress method, (2) MultipartMessage.addMessagePart and (3) MultipartMessage.setSubject are used to set the sections of our message. The message body of our MMS message is constructed in our MIDlet's getBody method as shown below.

Code sample 4

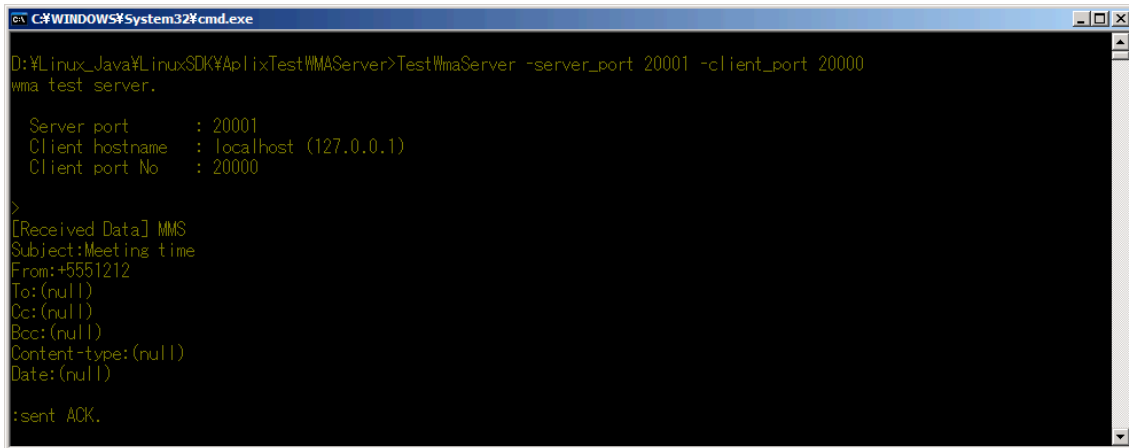
```

public MessagePart getBody() {
    String mimeType = "text/plain";
    String encoding = "UTF-8";
    String text = bodyTextBox.getString();
    byte[] contents = null;
    try {
        contents = text.getBytes( encoding );
    } catch ( java.io.UnsupportedEncodingException ex )
    { dbg( ex.toString() ); }

    // Create the MessagePart
    try {
        mBody = new MessagePart( contents, 0, contents.length, mimeType, "id1",
        null, encoding );
    } catch( javax.wireless.messaging.SizeExceededException ex )
    { dbg( ex.toString() ); }

    return mBody;
}

```

```
C:\WINDOWS\System32\cmd.exe
D:\Linux_Java\LinuxSDK\ApplixTest\WMA Server>TestWmaServer -server_port 20001 -client_port 20000
wma test server.

  Server port      : 20001
  Client hostname  : localhost (127.0.0.1)
  Client port No   : 20000

>
[Received Data] MMS
Subject:Meeting time
From:+5551212
To:(null)
Cc:(null)
Bcc:(null)
Content-type:(null)
Date:(null)

:sent ACK.
```

Figure 3: The WMA Test Server command window acknowledges receipt of the MMS message.

Summary

The WMA Test Server is a tool to help developers test messaging applications. It simulates the transmission of SMS, CBS and MMS messages to and from handsets to the messaging server. For troubleshooting the WMA Test Server, please refer to [FAQ #932](#) on [MOTODEV](#).

References

The Wireless Messaging API 2.0 By *C. Enrique Ortiz*, October 2005

<http://developers.sun.com/techtopics/mobility/midp/articles/wma2/>

Introduction of MMS in J2ME By *MOTODEV staff*, August 2006

<http://developer.motorola.com/?path=1.2.7.42.876>