LECTURE 11: SMS

http://dev.mobi/article/beyond-simple-xhtml-pages-pp-63-74

Mobile messaging, quite simply, is a far more prevalent mobile activity than viewing web content with a browser. To ignore messaging is to ignore the largest slice of the content consumer pie and a potentially profitable opportunity. Messaging comes in a variety of flavors: Short Messaging Service (SMS), which is text-only messaging; Multimedia Messaging Service (MMS) for sending photos, audio, video, and rich text in addition to plain text; Enhanced Messaging Service (EMS), a technology older and less capable than MMS, but marginally more capable than SMS.

Of particular interest as it relates to mobile web content is text messaging. More than 10 billion text messages are sent worldwide every month, and estimates for SMS support on handsets in use today range as high as 98%.1 In the U.K. alone, 41.8 billion text messages were sent in 2006.2 Because of its prevalence, person-to-person (one-to-one) text messaging is a familiar activity: A user types a message on her handset, sends it to a personal number, and the intended recipient receives the message on his handset. However, texting can also be a one-to-many or many-to-one relationship. Content providers big and small have leveraged the power of texting as a means of serving web content to mobile users in this way. For example, mobile users can conduct a web search by texting their search query rather than entering a query in a browser search field.

Google SMS (http://www.google.com/sms) offers SMS search, and the process combines texting and search query (see Figure 6-1). A user types keywords, e.g. "hotels san francisco," as a text message, sends the message to Google's SMS number (466453 in the U.S.), and search results are sent back as a text message. Only a few relevant results are sent, often as multiple text messages due to the recommended 160 character limit for SMS messages. Yahoo! oneSearch (http://mobile.yahoo.com/onesearch) and 4INFO (http://4info.net ) offer similar services, among others.

Google SMS search

Figure 6-1. Using Google SMS Search with a Nokia 6800 in my trusty Jeep Cherokee. (Yes, of course I was parked.)

PayPal (http://www.paypal.com ) has also embraced texting as a means for extending its popular web payment service to mobile phones. PayPal's Text To Buy (http://mobilewebbook.com/shorty/98098) allows consumers to send money via phone merely by texting an amount and email address to PayPal's SMS number, which is 729725 in the U.S. (see Figure 6-2). In addition to texting, PayPal Mobile Checkout (https://www.paypal.com/IntegrationCenter/ic\_mobile-checkout.html) offers PayPal's traditional payment services via mobile browser.

Paypal's Text to Buy

Figure 6-2. Sending money using PayPal's Text To Buy. (Nokia 6680)

SMS search and Text To Buy are only the tip of the proverbial texting iceberg. Imagine if the University of Texas directory (see chapter, Mobile Web Fundamentals) were also available by SMS. One could text "amy miller" to a number for the university and receive matching directory results by text message. The options are seemingly endless for employing text messaging to serve web content. Although I personally recommend you seek professional assistance if considering a custom messaging solution, DevelopersHome.com offers a very thorough SMS Tutorial at http://www.developershome.com/sms. A less replete but more digestible overview is How SMS Works by Howstuffworks.com, located at http://communication.howstuffworks.com/sms.htm.

The SMS numbers mentioned thus far--Google (466453), PayPal (729725)--are commonly known as Short Codes. They differ from phone numbers in that they act as a numeric domain name for text messaging, and they're often shorter in length than phone numbers, typically 4-6 digits. The numbers often map to letters (e.g. 82267 = "tacos"), much like toll-free advertising phone numbers. However, while Short Codes function irrespective of carrier or operator, they are not as universal as domain names because they are generally restricted to continents. So while 466453 is the Short Code for Google in the U.S., in the U.K. the Short Code may be an entirely different number. Further, because Short Codes are meant to be short, the quantity of available codes is limited. In North America, for example, Short Codes are five-digit numbers in the range 20000-99999, which results in 79,999 total numbers available.

To register a Short Code, visit one of the following:

\* U.S.: http://www.usshortcodes.com

\* U.K.: http://www.short-codes.com

Alternatively, if you are an organization based in the U.S., TextMarks (http://www.textmarks.com) offers a generic Short Code (41411) that can be used by any organization to send on-demand, customized messages to users, such as a web address, a marketing promotion, and so on.

BINARY SMS:

<http://dev.mobi/article/binary-sms-sending-rich-content-devices-using-sms>

Home

dotMobi certification

\* articles

\* forum

\* tools

\* blogs

\* ebooks

\* certification

\* training

\* showcase

\* MAG

Binary SMS: sending rich content to devices using SMS

Posted October 5th, 2007 by Julien Buratto

fire! This article will explain from a high level to a lower level how to write a simple SMS-enabled mobile application which will allow you to send “Over The Air installation messages”. After reading this article, you will be able to create a small J2ME applet which will theoretically send itself to another phone using an SMS message as the means of delivery. In fact, due to limitations in the Wireless Messaging API in J2ME (JSR 120/205) the port for WAP Push messages are “reserved” for security reasons. The ability to send content via the widely-supported SMS channel can be very useful for content providers.

What is an SMS ?

SMS messages are small number of packed bytes sent over the operator networks. Many of you have already experienced sending “text messages” from devices and many refer to sending SMS messages as “texting”.

A little technical introduction to SMS

The scope of this article is to be practical, so we will not go through all the technical and boring details of networks, but three concepts are pretty important do understand about sms.

\* SMS uses the concept of “port” just as a standard internet sockets do;

\* SMS messages have a limit in the body of 160 chars;

\* The body is not the only thing you can play with in an SMS, there is also the User Data Header.

Ports

When you hit a URL in your browser such as http://dev.mobi, you transparently call port 80 of a web server, by convention. The connection will be initialized on port 80 and then switched to a higher port to let other users access the same port of the web server. Port 80, as stated by IANA refers to the HTTP protocol, this means that a server, which is able to understand HTTP protocol request, will be awakened and will be ready to answer and process HTTP requests. The same is happens with SMS messages. You can send an SMS to a specific port of a phone and you will wake up a specific service on that device. Now, just as not all the computers have standard services (such as a web server), also not all mobile devices have services listening to ports. This is a very “manufacturer related” stuff, so you will need to check your phone what is enabled to accept.

Body and encoding

This could be a tough topic to treat but we will just describe very basic information which can be useful. SMS default encoding uses 7 bits to handle a character. This means that you can write in a SMS only those characters that are present on a very basic char table, yes those famous 127 chars. If you want to go more complicated stuff and send more “interesting” characters, then a group of 8 bits is needed and the table of available chars gets bigger. The available space is 1120 bits per SMS, no more, no less. You can have 160 chars using 7 bits or 140 chars using 8 bits

Note: The character “É” is included in the “basic table”, while the character “È” is in the “bigger tables”. If you write a message with the second char, then you will have less space available in the message. So, pay attention to which characters you are using when sending SMS ;)

User Data Header

The User Data Header (also known as UDH) is what a “high level developer” can set while to do something more than a simple “text message”. A UDH is very useful because you can send “invisible text messages” to mobile application (where to “mobile applications” I mean those running on mobile devices for example) or you can tell a device that the message will contain special information. It’s very similar to an XML file: you have to tell the parser what you are sending, and the content following the prolog which will be handled by the parser itself.

The UDH is mainly used to specify what ports our client (phone) will send the SMS to. It’s made by a set of hex number which describe:

<how long the UDH is> <the format used to specify ports numbers> <the port number length> <destination port number> <source port number>.

As a practical example, say I want to create a UDH to send a WAP Push, where the standard destination port for WAP pushes is 2948, the UDH will be:

06 05 04 0B 84 23 F0

where:

\* 06 means “hey the read the following 6 bytes”

\* 05 is the format for numbers, in this case hexadecimal numbers

\* 04 will tell the UDH that each port is represented using 4 character

\* 0B84 is the destination port, 2948 (decimal representation) or 0B84 (hexadecimal representation)

\* 23F0 is the source port, 9200 (decimal representation) or 23F0 (hexadecimal representation).

Note: Use a simple calculator to convert decimal numbers to hex; select “Dec”, put 2948 in the calculator, then press the button “Hex”. Any scientific calculator can do this and many operating systems have a little “calc” application somewhere which can do this.

Binary SMS

A “binary SMS” is an XML-formatted textual SMS which has been transformed with WBXML. WBXML is a “tag transformer”, this means that for each XML tag, a binary byte is associated. For example, the tag <SI> is converted as the binary character &#x0005;

Why WBXML?

Easy answer: the result of a WBXML transformation is smaller in the number of generated bytes than the verbose textual XML file itself.

Note: many tags are converted to bytes, but sometimes also contents (such as URL addresses) e.g the URL http://www.dev.mobi can be written in WBXML as 0Ddev.mobi , where “0D” stands for http://www.

“0C” is more generic and stands for http:// so you can write the URL in two ways:

0Ddev.mobi

or

0Cwww.dev.mobi

The first uses 9 chars (0D is one byte), the second 13 chars! I love WBXML :)

Terms

To be clear, by “WAP push” we do not mean a way to send SMS using WAP. What we mean is what is technically known as SI (Service Indication). A Service Indication is a binary SMS sent to a dedicated port on a device which informs the device that there is a URL waiting to be visited. The URL can be the address of a ring-tone in a webserver, the address of a JAD file to install a J2ME application or a JPG or simply a WAP page; it’s up to the “webmaster” of the webserver to put the contents on the web site, and it’s up to the device to understand how to handle the content. For more information on this aspect of the puzzle, please refer to the our Content Adapation series of articles.

Note: When I say “web server” I mean both web and WAP server since most web servers can be configured to deliver WAP content. By “OTA Configuration SMS” I mean a binary SMS which contains APN details (but can contain more) also known as “Internet Access Configuration”.

Note: There are special “Service Indication” messages which are normally called SL (Service Load) which are similar to “SI”. An SI asks the permission to the user before fetching the content over the network connection, SL download contents automatically without asking permission to the user. “SL” are very similar to MMS messages: the content is on a web server and the SMS tells the phone to download the message.

So Let’s Start

First of all, we need to read some papers (alas) to read some specifications on how to write XML content to be sent over SMS. A good start point could be the Nokia’s “Smart Messaging Specification 3.0.0” which can be downloaded from www.forum.nokia.com and also more technical docs from the Open Mobile Alliance (OMA) from www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

So our steps will be:

\* Decide what we want to send

\* Find the docs about that topic

\* Find the XML structure of the message to be sent

\* Customize the XML

\* Convert the XML to WBXML

\* Prepare the UDH

\* Send the UDH and the BODY

As we want to send a J2ME application to a new phone, then we need to send a “Service Indication” message, also known as “WAP push”. This “SI” can be used to push an SMS that will be read as a link to download something. Any content-type (image/jpeg, application/java-archive, …) is valid if the phone can open it.

So, who can tell which content-types can a device open ? WURFL can help us here! (see References)

Find the docs about that topic

Ok, let’s open the OMA link and lets go to the “Functional area” denoted “push”. Let’s get the “WAP Service Indication Specification “. In order to understand how does WBXML works, I would suggest to read paragraph 8.3.2 “Attribute Start Tokens”.

Find the XML structure of the message

For those of you that more like DTDs, look at section 7.2. All others can skip all the doc and go to the read example at section 9. The XML there is pretty self explanatory so I will do some customization directly.

<?xml version="1.0"?>

<!DOCTYPE si PUBLIC "-//WAPFORUM//DTD SI 1.0//EN" "http://www.wapforum.org/DTD/si.dtd">

<si>

<indication href=http://www.dev.mobi/is\_fun.html si-id="6532">

DevDotMobi is Fun !

</indication>

</si>

Convert to WBXML

Hex code Meaning

02 WBXML Version 1.2

05 SI 1.0 Public Identifier

6A Charset UTF-8

00 String table length = 0

45 <SI>

C6 <indication>

0C href="http://

03 String starts

\* 7777772E6465762E6D6F62692F69735F66756E2E68746D6C www.dev.mobi/is\_fun.html

00 String ends

11 si-id attribute

03 String starts

\*\* 36353332 6532

00 String ends

07 Action attribute (signal - medium)

01 Ends of attributes, now the content

03 String starts

\* 446576446F744D6F62692069732046756E2021 DevDotMobi is Fun !

00 String ends

01 </indication>

01 </SI>

\* These are strings used to pass contents to the SI, each character in the string is converted to its hexadecimal representation.

\*\* "6532" is to be considered a string of characters and not a number, so don't use the calculator to convert this number

Our body is, putting all the numbers together:

02056A0045C60C037777772E6465762E6D6F62692F69735F66756E2E68746D6C0

011033635333200070103446576446F744D6F62692069732046756E2021000101

(which is 130 chars)

Prepare the UDH

Preparing the UDH is pretty easy. Just start with “06 05 04” and then add the port numbers. WAP Push messages uses “destination port” 2948 while source port is 9200. Convert decimal port numbers to hexadecimal formats, so 2948 becomes 0B84 and 9200 becomes 23F0. Magically, the UDH is: 06 05 04 0B 84 23 F0

Send the SMS and the UDH

Now, what you need to do with this? Pretty simple, just put everything together and the SMS is ready to be sent.

<UDH> + <BODY>

UDH: 06 05 04 0B 84 23 F0

BODY: 02056A0045C60C037777772E6465762E6D6F62692F69735F66756E2E68746D6C0

011033635333200070103446576446F744D6F62692069732046756E2021000101

The complete message is then:

0605040B8423F0

02056A0045C60C037777772E6465762E6D6F62692F69735F66756E2E68746D6C0

011033635333200070103446576446F744D6F62692069732046756E2021000101

Which is 137 chars long (hey, it’s a binary SMS, Java uses UTF-8 encoding for binary messages, so the limit for 1 SMS is 140 chars, aren’t we cool ?)

The J2ME application

To be as practical as possible, we will need an environment to be used to send SMS messages and to describe the real world but as not everybody has access to an SMS gateway with UDHs and an interface where to enter binary code, we will be describing how to do the same with a J2ME application.

Disclaimer: As to WMA specifications, J2ME applications can’t access to special reserved ports to send binary SMS so unluckily the J2ME application in the phone will get a security exception and the message will not be delivered.

MIDP2.0 has two methods which enable to send messages: one requires a String, one requires an array of bytes. Yes, the first one is used to set text contents, the second one to set binary contents.

Text method: setPayloadText(String textualSMS);

Binary method: setPayloadData(byte[] binarySMS);

In order to be able to intall the J2ME application "Over The Air", compile it with your favourite J2ME environment (I like NetBeans with the Mobility Pack), copy the .JAD and .JAR files to your webserver and customize the URL in the Wap Push message.

Let’s now write a simple J2ME class that sends the message:

import javax.microedition.midlet.\*;

import javax.microedition.lcdui.\*;

import javax.wireless.messaging.\*;

import java.io.\*;

import javax.microedition.io.\*;

public class WapPush extends MIDlet {

// Connection to send a message

private MessageConnection conn;

public void startApp() {

// Put here your mobile number

String myMobileNumber="+39335...";

try {

// Create a binary message

// Destination port is expressed using the decimal notation

this.conn=(MessageConnection) Connector.open("sms://"+myMobileNumber+":2948");

// Specify that we are going to send a binary message

BinaryMessage bm=(BinaryMessage)conn.newMessage(MessageConnection.BINARY\_MESSAGE);

String ourContent="0605040B8423F0" +

"02056A0045C60C037777772E6465762E6D6F62692F69735F66756E2E68746D6C0" +

"011033635333200070103446576446F744D6F62692069732046756E2021000101";

bm.setPayloadData(convertHexToBinary(ourContent));

this.conn.send(bm);

// Let's throw exceptions

} catch (IOException ex) {

ex.printStackTrace();

} catch (Exception ex) {

ex.printStackTrace();

}

}

/\*\* This method comes from affinitystudios

\* read on http://discussion.forum.nokia.com/forum/member.php?u=56142

\* It converts Strings to data array of chars

\*/

private static byte[] convertHexToBinary(String hexData) throws Exception {

System.out.println("Length: "+(hexData.length()/2));

if ( hexData.length() % 2 != 0 ) {

throw new Exception("Must be an even number of hex digits" );

}

byte[] binaryData = new byte[hexData.length() / 2 ];

for (int i = 0; i < binaryData.length; ++i) {

String byteStr = hexData.substring(i\*2, i\*2+2);

int value;

try {

value = Integer.parseInt(byteStr,16);

} catch (NumberFormatException e) {

e.printStackTrace();

throw new Exception("");

}

binaryData[i] = (byte)value;

}

return binaryData;

}

public void pauseApp() {

}

public void destroyApp(boolean unconditional) {

}

}

Using an online SMS gateway

As J2ME blocks many binary messages, you can try an online gateway to send your binary messages. Many companies provide SMS gateway services, and most of them have an interface to send raw binary SMS messages (these are a “plus” to me) while some other companies just give you an interface to set the customized fields and make the binary stuff for you.

If you want to use a online SMS gateway, just remember to read their technical specifications. If you want to install yourself a SMS gateway which connects to external services (carriers/operators), I suggest Kannel, which is a free SMS gateway software and it is spread all over the world; Kannel likes to have the UDH and the body to be separated.

References

1. Reserved ports and more: J2ME “Wireless Messaging API” specifications

http://java.sun.com/products/wma/index.jsp

2. standard format for OTA contents: Open Mobile Alliance

http://www.openmobilealliance.org/tech/affiliates/wap/wapindex.html

3. How to send OTA configs, ring tones and operator logos: NOKIA

http://www.forum.nokia.com (seach for the “Smart Messaging Specification”)

4. How to detect a mobile phone: WURFL

http://wurfl.sourceforge.net

5. Sms gateway software: Kannel

www.kannel.org

Attachment Size

WapPush.java\_.txt 2.29 KB

Votes

Please rate this content:

Please rate this node. 1 = worst score, 10 = best score

+++++++++-

\* Add new comment

\* Delicious Delicious

\* Digg Digg

\* StumbleUpon StumbleUpon

\* Reddit Reddit

\* Technorati Technorati

\* Subscribe post

Search

natecow

\* News aggregator

\* Create content

\* My subscriptions

\* My account

\* Log out

Article Topics

Advertising Authoring CMS Content adaptation dev.mobi DeviceAtlas Device databases Device detection Framework Mobile AJAX Mobile design mobileOK MWI One Web Publishing ready.mobi Testing Tools W3C XHTML

more tags

Article Types

Case Study Feature Fixit Reference Resources Tip Tutorial

more tags

Recent popular content

\* Home (696)

\* DotMobi Mobile Web Developer's Guide (106)

\* Site Templates (98)

\* Introduction to WURFL (79)

\* dotMobi Mobile Web Developer Certification (68)

more

Active in the forum

\* Some problems with agent (browser) detection

\* 3G Icons

\* google analytics for mobile

more

DotMobi news

\* dotMobi Happenings: More happening.

\* .mobi sites = iPhone web apps = sites for all phones

\* dotMobi Happenings: They're happening.

\* Say hello to world.mobi

\* What a difference a year makes ...

more

© 2007 mTLD, Ltd. All rights reserved. Legal

<http://www.onjava.com/pub/a/onjava/2004/06/09/sms.html?page=2>

ONJava.com -- The Independent Source for Enterprise Java

O'Reilly Network.oreilly.comSafari Bookshelf.Conferences.

Sign In/My Account | View Cart

Articles Weblogs Books School Short Cuts Podcasts

advertisement

Click here to find out more!

JavaOne(SM) Conference: There's still time to save $100. Register today!

AddThis Social Bookmark Button

Listen Print Discuss Subscribe to ONJava Subscribe to Newsletters

SMS-Powered Applications SMS-Powered Applications

by Dejan Bosanac

06/09/2004

Every business tries to get as close as it can to its customers. The aim is to keep customers up to date with company news, products or service updates, relevant information about their accounts, or to send them notifications for important events. Email messages are widely used for these purposes and that is usually an appropriate way to inform customers. But there is a drawback to using email exclusively for communication with customers. Not all customers have a habit of checking their mailboxes regularly, so information can be delivered in a matter of hours or even days. Sometimes, this is not fast enough, and in some cases, it could be good for the company's public image to make it better. Mobile phones are a commodity today, and SMS (Short Message Service) messages offers an intriguing solution. Messages are delivered in a matter of seconds, and customers are far more likely to receive and read them, no matter their Internet habits or current locations.

This is by no means the only way SMS messages could work for you. You could provide your users with two-way communication by enabling customers to send you an SMS request in a specified format and get a response from the system. This user interface would be far from being user-friendly, but for simple request-response situations, it would work well enough.

Beside SMS messages, there are many technologies currently used for messaging by mobile users. Still, SMS messaging is the most widely adopted, and there are good reasons for that. MMS (Multimedia Messaging Service) messages have become popular, but many mobile devices doesn't support them yet, and some people are unlikely to change their current phones for several years. EMS (Enhanced Message Service) is a proprietary technology, which restricts its wide adoption among phone makers. In this light, some analysts predict that SMS will be dominant technology in this field for the next few years. So we can definitely say that it is here to stay.

There are many companies that offers SMS gateway services and enables various interfaces (HTTP, FTP, XML-RPC, SMTP) that could be used for sending (and in some cases, receiving) SMS messages. Also, you could find some gateway products and open source projects for the same purpose. This approach is appropriate for most cases, because developers know these technologies well and it doesn't require a big development effort for a solution that will work. But if you want to heavily use SMS messages or greater flexibility in your system, you will probably want to skip the "middleman" and work directly with your mobile provider.

This article will describe the basics needed for developing an SMS-powered application that can communicate directly with the mobile operator of your choice, all using Java and open source tools and libraries.

Protocol

SMS messages are transferred using Short Message Peer to Peer (SMPP) protocol. This is an open industry standard protocol developed to enable transfer of short messages between mobile users, called External Short Message Entities (ESMEs), and Short Message Service Centers (SMSCs). An ESME represents a fixed-network SMS client, and that is where our application resides in the system. For the SMSC, we will assume a mobile provider or a center that is responsible for delivering the message to mobile user. The communication may also involve Routing Entities (REs) that are responsible for message routing between ESMEs and SMSCs (and of course, other routing entities).

SMPP defines a set of operations in the form of Protocol Data Units (PDUs). PDUs are well-formed packets that are transferred between entities. A PDU consists of a header and an optional (depending on the PDU type) body. We will not dig deeper into PDU formats in this article because all of these low-level operations are abstracted by the API that will be described later in the text. Detailed description of the SMPP protocol could be found on the official SMPP forum site.

From the OSI stack perspective, SMPP presents an application layer protocol and relies on a TCP/IP (or X.25) network protocol that is responsible for packet delivery. So the basic requirement for using SMS messages is providing a reliable network connection to your mobile provider (just as it would be needed for any HTTP data transfer).

The first thing the ESME (our application) should do is to establish a session with the message center. There are three connection types that could be used, depending on the application's role in the system. An ESME can establish the connection as a transmitter (TX), meaning it can only send messages to the SMSC for a delivery to mobile users. Next, it can behave as a receiver (RX), in which case it can only receive messages from an SMSC. Finally, first two modes are combined in a transceiver (TRX) session type that enables application to both send and receive SMS messages.

In order to get a picture of how a SMPP session works, we will go through one example where the application connects to the SMSC as a transceiver, gets a request from the user (such as a request for the balance on a user's account for an online banking system), and responds back (with the user's balance). This arrangement is shown in Figure 1. This will also be our example Java application later. The labels on the arrows represents PDU types that are exchanged between an ESME and a SMSC.

Figure 1

Figure 1. Example of PDUs transferred between an ESME and a SMSC

After establishing the network connection and binding as a transceiver, the ESME is enabled to receive messages. The user sends SMS messages in an appropriate format (such as BALANCE 123456, meaning "Give me the balance for account number 123456") and the SMSC delivers the message to the application (shown as the deliver\_sm and deliver\_sm\_resp PDUs). After checking for correct format and retrieving the data from the database, the application returns the current balance back to the user (the submit\_sm and submit\_sm\_resp PDUs). The application can exchange messages with the message center as long as it has a valid, bound session to it. When the application wants to terminate the session, it should unbind the session (with the unbind and unbind\_resp PDUs) and close the network connection.

A few more things should be said about the SMPP protocol. First, it is an asynchronous protocol, which means that the ESME could send the next message before it receives a confirmation that the previous one had been received by message center. This approach leads to better network utilization and overall performance. The asychronous nature of the protocol also affects the API architecture, which will be discussed in the later section.

Security is always a concern when one is talking about communication protocols. PDUs are transferred plain, so they are vulnerable to interception and impression attacks. If you want to prevent these kind of attacks, you should consider some additional security measures, such as using a leased line to the message center or using an SSL connection.

API

One way to make your application "speak" SMPP is to use the SMPP API, an open source Java API. We will demonstrate the basics of this library through a simple demo application.

For a start, let's define requirements for this demo application. It will serve as an extension of the e-banking system, enabling customers to check their balances using their mobile phones. For each customer there is a defined phone number that he or she must use for communicating with the system. The format of the user's request message is:

BALANCE <ACCOUNT\_NUMBER>

where BALANCE is a keyword and is followed by the account number of the customer. If the user makes a proper request, he or she will receive a message containing the current balance. Otherwise, an error message with an appropriate explanation will be delivered to the phone. As a technical requirement, we will insist that the application can successfully deal with network connection problems, because for this kind of application, needing to restart every time the connection to the SMSC is lost would make the application much less usable and hard to administer.

For a start, we have to establish a connection to the SMSC and bind it with appropriate credentials. For that purpose, we should use the ie.omk.smpp.Connection class. As noted earlier, SMPP is an asynchronous protocol, but the nature of the protocol does not keep us from using it synchronously, if we desire. SMPP API enables communication of both types, with synchronous communication as the default. In our demo we will still use it in asynchronous way because of all of the benefits mentioned above. Here is the code snippet for establishing the connection to the SMSC:

private void connect() {

try {

conn = new Connection("localhost", 2775, true);

conn.addObserver(this);

} catch (UnknownHostException uhe) {

System.exit(0);

}

boolean retry = false;

while (!retry) {

try {

conn.bind(

Connection.TRANSCEIVER, "sysId",

"secret", null

);

retry = true;

} catch (IOException ioe) {

try {

sleep(60 \* 1000);

} catch (InterruptedException ie) {

}

}

}

}

All example code for this article can be downloaded at the end of the article.

Pages: 1, 2

Next Pagearrow

Resource Management: A Critical Look at RAII Michael Feathers

The Logs That Bind.. Michael Feathers

Outside the Design Boundary Michael Feathers

Can a program lie to you the way a story or essay can? Andy Oram

> More from Beautiful Code

Search ONJava

Tagged Articles

Post to del.icio.us

This article has been tagged:

\* java

\* mobile

\* sms

java

Articles that share the tag java:

Profiling Your Applications with Eclipse Callisto (113 tags)

What Is a Portlet (101 tags)

Parsing an XML Document with XPath (79 tags)

The REST of the Web (75 tags)

Eclipse Plugins Exposed, Part 1: A First Glimpse (69 tags)

View All

mobile

Articles that share the tag mobile:

Roll Your Own Series 60 Phone Applications (12 tags)

Python on Your (S60) Phone (11 tags)

Developing Wireless Content using XHTML Mobile (10 tags)

Using Mobile Phones to Model Complex Social Systems (10 tags)

Developing Web-Service-Driven, Smart Mobile Applications (9 tags)

View All

sms

Articles that share the tag sms:

SMS-Powered Applications (2 tags)

View All

Sponsored Resources

\* Inside Lightroom

Recommended for You

How to Build an SMS Service

How to Build an SMS Service

Retford, Schwartz, 1st Edition $9.99

Java Message Service

Java Message Service

Monson-Haefel, Chappell, 1st Edition $34.95

Introduction to HTML and CSS

Introduction to HTML and CSS

Online, Self-paced Course - 1:1 Coaching, Free Book. Get Certified Experience! $298.00

Advertisement

Click here to find out more!

Sponsored by:

Contact Us | Advertise with Us | Privacy Policy | Press Center | Jobs

Copyright © 2000-2008 O'Reilly Media, Inc. All Rights Reserved. | (707) 827-7000 / (800) 998-9938

All trademarks and registered trademarks appearing on the O'Reilly Network are the property of their respective owners.

For problems or assistance with this site, email help@oreillynet.com

ONJava.com -- The Independent Source for Enterprise Java

O'Reilly Network.oreilly.comSafari Bookshelf.Conferences.

Sign In/My Account | View Cart

Articles Weblogs Books School Short Cuts Podcasts

advertisement

Click here to find out more!

JavaOne(SM) Conference: There's still time to save $100. Register today!

AddThis Social Bookmark Button

Listen Print Discuss Subscribe to ONJava Subscribe to Newsletters

SMS-Powered Applications

Pages: 1, 2

First we have to instantiate a new Connection object with the desired host and port parameters. The third parameter defines the connection as asynchronous and could be left out if you want to use some other approach.

Next, we should bind the connection with the connection type and credentials provided by the message center. We will bind as a TRANSCEIVER, as we want to both submit and receive messages. The parameters that have to be passed are system identification and a "secret," used for authentication purposes. Also, you should pass a "system type" parameter that defines the application's purpose, such as VMS for a voice mail system. This parameter is not required by all SMSCs, and its default value is null.

At this point, the underlying network connection will be established, and if the SMSC is currently down or unreachable, an IOException will be thrown. There are a few things that you could do in this case, such as log the error and exit. Our requirement is to deal with these situations, so we will try to bind again after a one minute delay.

public void run() {

while (!exit) {

connect();

synchronized(this) {

try {

wait();

} catch (InterruptedException ie) {

}

}

}

}

Our main application thread is quite simple: it binds the connection and sleeps until the connection terminates or an exit flag is set. In the first case, we will try to bind again (see the connect method above). In the latter case, we exit the application due to some unrecoverable error such as when binding is refused from the SMSC (which means that the submitted parameters are not correct) -- in this case, there is no point executing the application until the error is resolved. This is of course only one possible architectural approach, and you should adopt this API to your needs and environment.

One more tip that could be used is to create a shutdown hook, which will unbind our application from the SMSC when the application terminates or crashes. That way, we will be sure that we have done everything to write an application that will behave properly under any circumstances. The code below simply unbinds the connection if it exists and is currently bound:

package net.nighttale.smsdemo;

import ie.omk.smpp.Connection;

public class Hook extends Thread {

public void run() {

System.out.println("Unbinding");

Connection conn = SMSDemo.getInstance().getConnection();

if(conn != null && conn.isBound())

conn.force\_unbind();

}

}

Adding it to the list of runtime hooks is a simple call:

Runtime.getRuntime().addShutdownHook(new Hook());

One issue with asynchronous communication is how to implement receiving functionality. The observer pattern is the right solution for these sort of problems, so the SMPP API provides a ConnectionObserver interface for this purpose. To be enabled to receive messages (and other relevant events), our demo class has to implement this interface, and we have to write code for two methods defined by it. Next, this observer should be attached to the connection using the addObserver method, as shown in the in the connect method above. The first method we will look at is update, which accepts Connection and SMPPEvent as arguments.

public void update(Connection conn, SMPPEvent ev) {

if (ev.getType() == SMPPEvent.RECEIVER\_EXIT

&& ((ReceiverExitEvent)ev).isException()) {

synchronized(this) {

notify();

}

}

}

There are three types of events generated by the receiver thread. One caught in the demo is SMPPEvent.RECEIVER\_EXIT, which is generated when receiver thread exits either normally or due to an exception. In the latter case, the isException() method will return true, which means that the network connection to the SMSC is invalid and we should reconnect in order to continue. In this case, we wake up our main thread and try to make a new connection. The second event type is SMPPEvent.RECEIVER\_EXCEPTION, which means that receiver has caught an exception that has not caused it to terminate. It would be useful to log this event, but to keep the example code simple, this logging has been left out. Finally, there is SMPPEvent.RECEIVER\_START, which is generated when the thread receiver starts, and can be ignored.

The more important method of the ConnectionObserver interface is packetReceived:

public void packetReceived(Connection conn,

SMPPPacket pack) {

switch(pack.getCommandId()) {

case SMPPPacket.DELIVER\_SM :

try {

SubmitSM response = processRequest(pack);

SubmitSMResp smr =

(SubmitSMResp)conn.sendRequest(response);

} catch (Exception e) {

e.printStackTrace();

}

break;

case SMPPPacket.BIND\_TRANSCEIVER\_RESP :

if (pack.getCommandStatus() != 0) {

System.out.println("Error binding: " +

pack.getCommandStatus());

exit = true;

synchronized(this) {

notify();

}

} else {

System.out.println("Bounded");

}

}

}

This is the point where all the PDUs sent from the SMSC are collected in the application. In this demo, we will collect only two types: SMPPPacket.DELIVER\_SM (a SMS submitted by the customer) and SMPPPacket.BIND\_TRANSCEIVER\_RESP (a response PDU from the SMSC to our bind request). For the binding response, if we detect that we haven't successfully bound to the SMSC, we set the exit flag, which will terminate the application.

Before we proceed to the SMS processing, it should be noted that implementing the "raw" ConnectionObserver interface leads to clumsy code, because you will have a long, ugly switch-case structure in these two methods. I used it in this demo for demonstration purposes, and I didn't handle all that has to be handled for a production application. So, if you will be coding an application that will handle (nearly) all possible cases, you should consider using the ie.omk.smpp.event.SMPPEventAdapter class, which implements this interface and has predefined methods for all packets and events that could be handled. You can learn more about this class by reading the SMPP API's Javadoc.

OK, now let's get back to our demo. All SMS messages delivered from the SMSC are handled by the processRequest method.

private SubmitSM processRequest(SMPPPacket request)

throws BadCommandIDException {

SubmitSM sm =

(SubmitSM)conn.newInstance(SMPPPacket.SUBMIT\_SM);

sm.setDestination(request.getSource());

String[] parts = request.getMessageText().split(" ");

logPacket(request, "IN");

if (parts[0].equalsIgnoreCase("balance")) {

User user =

User.findByPhone(request.getSource().getAddress());

if (user == null)

sm.setMessageText("Your phone number is not " +

"registered in our database! " +

"Please contact one of our offices");

else if (!user.getAccountNumber().equalsIgnoreCase(parts[1]))

sm.setMessageText("Account number that you " +

"have entered is not correct! " +

"Please try again");

else

sm.setMessageText("Balance on your account is "

+ user.getBalance() + "$");

} else {

sm.setMessageText("Wrong message format! " +

"Please send BALANCE " +

"<ACCOUNT\_NUMBER>");

}

logPacket(sm, "OUT");

return sm;

}

Here, we will create a response message that will be delivered to the user by calling the newInstance method of the Connection class with the appropriate PDU type (SUBMIT\_SM in this case). Next, we should specify the phone number of the user by setting the destination field of the packet (setDestination), which, in the case of a simple reply, is exactly the same as the source of the received message. Now, we parse the text of the message to get all of the necessary data to process the input. Thus far, we have been discussing and coding communication, but this is the entry point to the business logic of the application. For this simple demo, I have implemented a mock-up User class to represent the business logic.

package net.nighttale.smsdemo;

public class User {

private String phoneNumber;

private String accountNumber;

private float balance;

public User (String phoneNumber,

String accountNumber,

int balance) {

this.phoneNumber = phoneNumber;

this.accountNumber = accountNumber;

this.balance = balance;

}

public static User findByPhone(String phoneNumber) {

if (phoneNumber.equals("063123456"))

return new User(phoneNumber, "123456", 10000);

else

return null;

}

public String getAccountNumber() {

return accountNumber;

}

public float getBalance() {

return balance;

}

public String getPhoneNumber() {

return phoneNumber;

}

public void setAccountNumber(String accountNumber) {

this.accountNumber = accountNumber;

}

public void setBalance(float balance) {

this.balance = balance;

}

public void setPhoneNumber(String phoneNumber) {

this.phoneNumber = phoneNumber;

}

}

The class has one finder method (findByPhone), a constructor, and getter/setter methods. For simplicity, findByPhone returns a valid user only for one phone number and null in all other cases. Of course, this should be replaced with a real class implemented in any convenient technology, such as EJBs, that would search and retrieve data from the database.

Depending on the message format, the user's telephone number, and (if the message was well-formatted) the submitted account number, some appropriate text is set as the return message to the user, using the setMessageText method. At the end, the message is submitted to the SMSC using the sendRequest method of the Connection class and passing the previously formed PDU as a parameter to it.

Simulator

When you are developing a SMPP-based application, you will probably want to set up a local environment for development and initial testing purposes. This is often useful because the process of obtaining access to an operator's SMSC could be time-consuming, and providers usually run detailed acceptance tests before letting your application access their SMSC. For all of these reasons, it is convenient to install some kind of SMSC simulator in your development environment. In this article, we will briefly describe SMPPSim, an open source Java SMSC simulator, and test our demo application with it.

After downloading and extracting the archive, you are ready to run the simulator by simply typing

startsmppsim

on a Windows platform, or

./startsmppsim

on Unix-like systems, in the root directory of the extracted simulator.

By default, the SMPPSim will open a network connection on port 2775 and wait to be bound by our application.

Configuration of SMPPSim is done through one of the props files in the conf directory. Depending on the underlying operating system, you should use the props.unix or props.win file. We will not dig deeper into setting details, and instead just use it to turn on message delivery from the SMSC to our application. To do that, you should change settings in the # DELIVERY SERVICE section. You should set the number of messages that message center will send to our application per minute with the parameter DELIVERY\_MESSAGES\_PER\_MINUTE and the file that contains messages that will be randomly sent (DELIVER\_MESSAGES\_FILE). The default message delivery file is deliver\_messages.csv, which can be found in the root folder of SMPPSim. Now let's make a few test cases for the demo.

063123456,1000,A test message

063123456,1000,BALANCE 123456

063123456,1000,balance 123456

063123456,1000,balance 123457

063123457,1000,balance 123456

In the delivery file, each line represents one SMS message to be delivered to the application. You should specify the user's phone number, ESME system identification, and text message, in a comma-separated fashion. Messages will be delivered to the application in random order at the rate defined in the configuration file. Recall that the findByPhone method of the User stub class will return a valid user only if the specified number is 063123456. The appropriate account number for that phone number is 123456. So this delivery file should generate all possible test cases. Set the number of messages to be delivered per minute (for example, 10), copy the delivery file (or generate one of your own) from the demo folder to SMPPSim folder, run SMPPSim, run the demo application, and watch how they communicate. You should get something like this on your console window.

2004-04-04 02:51:41,585 DEBUG [Thread-1] ie.omk.smpp.util

Loaded API properties from /smppapi.properties

2004-04-04 02:51:41,585 DEBUG [Thread-1] ie.omk.smpp.util

-- listing properties --

smppapi.net.tcp.so\_timeout=300000

smppapi.net.autoflush=true

smppapi.connection.rcv\_daemon.ioex\_count=3

Bounded

IN: 063123456 - BALANCE 123456

OUT: 063123456 - Balance on your account is 10000.0$

IN: 063123456 - balance 123457

OUT: 063123456 - Account number that you have entered is not

correct! Please try again

IN: 063123456 - balance 123456

OUT: 063123456 - Balance on your account is 10000.0$

IN: 063123456 - BALANCE 123456

OUT: 063123456 - Balance on your account is 10000.0$

IN: 063123456 - A test message

OUT: 063123456 - Wrong message format! Please send BALANCE

<ACCOUNT\_NUMBER>

IN: 063123456 - balance 123456

OUT: 063123456 - Balance on your account is 10000.0$

IN: 063123457 - balance 123456

OUT: 063123457 - Your phone number is not registered in our

database! Please contact one of our offices

IN: 063123456 - balance 123456

OUT: 063123456 - Balance on your account is 10000.0$

IN: 063123456 - balance 123457

OUT: 063123456 - Account number that you have entered is not

correct! Please try again

IN: 063123457 - balance 123456

OUT: 063123457 - Your phone number is not registered in our

database! Please contact one of our offices

IN: 063123456 - BALANCE 123456

OUT: 063123456 - Balance on your account is 10000.0$

IN: 063123456 - balance 123457

OUT: 063123456 - Account number that you have entered is not

correct! Please try again

Now you can try to start and stop the simulator (to simulate network problems) and see how the application reacts.

Further Research

We've scratched the surface of the SMPP protocol and some ways you can use it on your project. There are many details that were left out, but I hope that you are at least a little curious and that you will dig deeper into this topic. To do that, you should follow one of the related links. SMS messages are definitely a powerful way to provide instant message delivery to a large audience. We mentioned just a few possible usages here, but I'm sure that more and more applications will be opened through this interface to their users.

However before starting to code your SMPP solution, check some of the existing gateway projects and products (such as Kannel). It may save you time if your requirements are not complex, and you certainly don't want to reinvent the wheel and make your own gateway if you don't need to.

Related Links

\* SMSDemo.zip: sample code for this article

\* SMPP Forum: SMPP specification, forum, links, tools

\* SMPP API: Open source SMPP library for Java

\* SMPPSim: Open source SMSC simulator

\* Kannel: Open source WAP and SMS gateway

\* SMS Tools: SMS-related tools, including another Java API

Dejan Bosanac is a software developer, technology consultant and author. He is focused on the integration and interoperability of different technologies, especially the ones related to Java and the Web.

Return to ONJava.com.

Even simple phones can do SMS. Does it have a place in your mobile app?

You must be logged in to the O'Reilly Network to post a talkback.

Post Comment

Full Threads Oldest First

Showing messages 1 through 12 of 12.

\* I need complete details about SMPP

2007-11-29 22:40:25 SMPPprotocaid [Reply | View]

Sir I am trying to understand & develop an SMSC software.Your software includes all the necessary constraints.I am getting problem to understand the flow of your software.If you can help me then please send the flow of your software in a sequence so that I can start working on it.I shall be grateful to you for this kind of favour.

Note:what are the exact requirements need for simply sending sms by using smpp as a software and

configuration setup.

And I download your SMSDemo but how can i run please send me details.I tried with ant it showing so many errors lastly build fail.

Thanking You

o I need complete details about SMPP

2007-11-29 22:41:49 SMPPprotocaid [Reply | View]

Sir I am trying to understand & develop an SMSC software.Your software includes all the necessary constraints.I am getting problem to understand the flow of your software.If you can help me then please send the flow of your software in a sequence so that I can start working on it.I shall be grateful to you for this kind of favour.

Note:what are the exact requirements need for simply sending sms by using smpp as a software and

configuration setup.

And I download your SMSDemo but how can i run please send me details.I tried with ant it showing so many errors lastly build fail.

Thanking You

nareshbodduluri@gmail.com

\* Regarding SMSC

2007-04-25 05:38:19 somsuresh [Reply | View]

Sir i have a query like if i m using your smsdemo code to send sms i just want to know who is going to provide me the SMSCs service and how do i get the connection,userid,and password .Well I also want to know what would be the cost of this Service as well as for sending sms to mobile.Kindly reply back as early as possible..

With Regards,

Suresh.

\* help

2006-10-15 02:58:32 nazari [Reply | View]

hi

i have downloded the smsdemo but i can't use it

because it hasn't an exe file and i don't know how to use from it and connect it to smppsim.

please help me

thanks alot

\* How can i send MMS from pc to handphone?

2006-09-27 23:42:47 Apit [Reply | View]

i currently doing a project using labview....but i had a problem finding a program using java for sending a picture from my PC to handphone.......can someone pls send me the program as soon as possible???...i need it urgently.....tanx....

apitrii@hotmail.com

\* How can i send MMS from pc to handphone?

2006-09-27 23:42:41 Apit [Reply | View]

i currently doing a project using labview....but i had a problem finding a program using java for sending a picture from my PC to handphone.......can someone pls send me the program as soon as possible???...i need it urgently.....

\* Please help me

2006-08-02 00:29:53 prabhatidubey [Reply | View]

Sir I am trying to understand & develop an SMSC software.Your software includes all the necessary constraints.I am getting problem to understand the flow of your software.If you can help me then please send the flow of your software in a sequence so that I can start working on it.I shall be grateful to you for this kind of favour.

Thanking You

My email id is

prabhati.dubey@3i-infotech.com

\* how to hire SMSC service

2006-05-03 05:32:00 VikramRawal [Reply | View]

I want to register and connect to real SMSC service provider. But I am not able to find any live SMSC service site. Please help me.

\* Change the smsc configuration

2006-04-19 03:26:16 theDistiller [Reply | View]

Hi (excuse my english i m french)

I m new in smsc programming , and i would like to know if it is possible to configure the smsc server in order to add a port number in the optionnal header of the sms network packet which are send to our mobile phone.

Thanks

\* smsc

2006-04-01 22:35:12 iamlateef [Reply | View]

great article but i need a little help on the smsc side. How do one go about securing an account with an smsc ?

thanks

lateef...iamlateef@gmail.com

\* SMPP accounts are usually expensive

2006-02-04 10:01:50 marre99 [Reply | View]

If you only need to send a few SMS you don't need a SMPP server. Just search for "bulk sms http" on google and you will find several SMS gateways that provide an HTTP api.

regards,

Marre http://smsj.sourceforge.net/

\* I need your help

2005-05-20 16:51:10 SureshR [Reply | View]

Sir I am trying to understand & develop an SMSC software.Your software includes all the necessary constraints.I am getting problem to understand the flow of your software.If you can help me then please send the flow of your software in a sequence so that I can start working on it.I shall be grateful to you for this kind of favour.

Thanking You

My email id is

nadeem.a@agastya.in OR

jhanavnit\_2001@rediffmail.com

Resource Management: A Critical Look at RAII Michael Feathers

The Logs That Bind.. Michael Feathers

Outside the Design Boundary Michael Feathers

Can a program lie to you the way a story or essay can? Andy Oram

> More from Beautiful Code

Search ONJava

Tagged Articles

Post to del.icio.us

This article has been tagged:

\* java

\* mobile

\* sms

java

Articles that share the tag java:

Profiling Your Applications with Eclipse Callisto (113 tags)

What Is a Portlet (101 tags)

Parsing an XML Document with XPath (79 tags)

The REST of the Web (75 tags)

Eclipse Plugins Exposed, Part 1: A First Glimpse (69 tags)

View All

mobile

Articles that share the tag mobile:

Roll Your Own Series 60 Phone Applications (12 tags)

Python on Your (S60) Phone (11 tags)

Developing Wireless Content using XHTML Mobile (10 tags)

Using Mobile Phones to Model Complex Social Systems (10 tags)

Developing Web-Service-Driven, Smart Mobile Applications (9 tags)

View All

sms

Articles that share the tag sms:

SMS-Powered Applications (2 tags)

View All

Sponsored Resources

\* Inside Lightroom

Recommended for You

How to Build an SMS Service

How to Build an SMS Service

Retford, Schwartz, 1st Edition $9.99

Java Message Service

Java Message Service

Monson-Haefel, Chappell, 1st Edition $34.95

Introduction to HTML and CSS

Introduction to HTML and CSS

Online, Self-paced Course - 1:1 Coaching, Free Book. Get Certified Experience! $298.00

Advertisement

Click here to find out more!

Sponsored by:

Contact Us | Advertise with Us | Privacy Policy | Press Center | Jobs

Copyright © 2000-2008 O'Reilly Media, Inc. All Rights Reserved. | (707) 827-7000 / (800) 998-9938

All trademarks and registered trademarks appearing on the O'Reilly Network are the property of their respective owners.

For problems or assistance with this site, email help@oreillynet.com